The Multilingual Lion:* TEX learns to speak Unicode

Jonathan Kew
SIL International

January 11, 2005

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 focus on XETEX, a system that extends TEX with direct support for modern OpenType and AAT 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. XETEX (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, Text Encoding Converter) and auxiliary libraries (ICU, TECkit). Thus, it illustrates how such components can be leveraged to provide the benefits of Unicode within an existing software system.

This paper should be of interest to those involved in multilingual and multiscript publishing, as well as developers seeking to enhance legacy systems to take advantage of the benefits of Unicode. The merger of legacy and Unicode-based technologies means that the benefits of many years of development in the TEX world become available for document production in a much wider range of languages.

Some background familiarity with TEX may be helpful, but the paper's focus will be on the integration of Unicode technologies, not on technical details of TEX itself. A general awareness of encodings, complex scripts, and font technologies will be assumed.

1 Background

The TEX typesetting system has a 20-year history as a stable and reliable tool for producing well-formatted documents from marked-up source text, and offers a great deal of power, flexibility and extensibility by virtue of a powerful macro language. The extensive user community, especially in the academic world, has created a large collection of supporting packages for many different types of document.

For those unfamiliar with TEX, a brief overview may be helpful. The TEX processor reads a source document, recognizing characters as either text to be typeset or markup according to scanning rules and (customizable) character categories. It expands macros and executes commands (setting parameters

*Why a multilingual *lion*? Because TEX's logo is a lion; see Knuth's *The TEXbook* (Addison Wesley: 1984) or other sources.

Source text	*Typeset result*	*Notes*
`\'{a}`	á	`\'` is one of various commands to add an accent to a letter
`\c{c}`	ç	an accent that attaches below the letter
`\aa`	å	special command for a specific character
`---`	—	implemented as a ligature in standard TEX fonts
`\alpha`	α	one of many symbols available in *math mode*
`{\dn acchaa}`	अच्छा	requires use of a special preprocessor and custom fonts

Figure 1: Traditional TEX input conventions for non-ASCII characters in 7-bit source text.

to control the typesetting process, for example), and forms the text into paragraphs and pages. Finally, a compact representation of the typeset pages is written out to a DVI ("device independent") file; a subsequent device driver process reads this `.dvi` file and renders the pages to a specific destination such as a screen or printer. (In the case of XHTEX, to be discussed below, this output format has been extended and renamed XDV, and the default behavior is to automatically run an XDV-to-PDF processor, so that the effective output format is PDF.)

Over the years, TEX has been used with many non-English languages, often using combinations of custom-encoded 8-bit fonts and different input encodings and conventions. However, TEX's roots are unquestionably in Latin-script typography; the system originally processed 7-bit text (usually ASCII), accessing 8-bit fonts for output. Version 3 extended the system to support 8-bit input text, and provided some enhancements for multilingual use, but support for many non-Latin and complex scripts remains a problem.

In addition to standard 8-bit codepages, there are ways of using TEX's programmability to allow input of additional text elements as by representing them as character sequences. Some conventions are so widely used that many users think of them as a standard part of the TEX program (though this is not really the case); others are associated with macro packages for particular languages; and still others are created just for specific projects. Figure 1 shows a few examples of typical TEX input conventions for non-ASCII characters.

While these conventions can be extended almost indefinitely, they tend to clutter the source text; and they rely on a variety of custom-encoded fonts to provide all the symbols needed. Unicode offers the possibility of a far simpler model for typesetting multilingual text, where each character needed is represented in the source not by some sequence of commands, but as itself.

In the case of complex scripts such as Devanagari, solutions based on standard TEX typically involve a custom preprocessor that performs the contextual analysis needed for proper rendering of the script, starting from some (often romanized) input convention, and emits special TEX commands to access the appropriate glyphs from custom 8-bit fonts. While such solutions can work, they may be complex to use, and fragile in how they interact with various other macro packages for document formatting. And trying to combine several such solutions to create a highly multilingual document, using several complex scripts simultaneously, goes far beyond what typical users can be expected to achieve.

To address these issues, an extended version of TEX known as XHTEX has been developed. This is a Unicode-based multilingual typesetting system that works with existing "smart font" technologies to provide complex script support, within the framework of the formatting power, flexibility, and programmability of TEX.

2 Examples of use

Before looking at what was involved in extending TEX to support Unicode and smart font technologies for text rendering, I will show a few brief examples of XƎTEX at work. Figures 2 to 5 illustrate how readily Unicode text now fits into the TEX typesetting paradigm. In each case, the "raw" source (text and markup) is shown alongside the typeset result.

```
\halign{#\hfil\quad&#\hfil\cr
\bf UCA default&\bf Tailoring:
 \& Đ < dž  <<< Dž  <<< DŽ\cr
dan&    dan\cr
dubok&  dubok\cr
džabe&  đak\cr
džin&   džabe\cr
Džin&   džin\cr
đak&    Džin\cr
Evropa& Evropa\cr}
```

UCA default	**Tailoring:** & Đ < dž <<< Dž <<< DŽ
dan	dan
dubok	dubok
džabe	đak
džin	džabe
Džin	džin
đak	Džin
Evropa	Evropa

Figure 2: From a presentation about collation, using extended Latin letters. In standard TEX, this could be done using control sequences to encode the accented letters, but XƎTEX handles the Unicode text directly.

```
\font\han="STSong:color=660000" at 16pt
\font\rom="Gentium:color=000066" at 8pt
\def\hc#1#2{\vtop{\hbox{\han #1}
 \hbox{\kern10pt\rom #2}}}
\vtop{\hc{書く}{ka-ku}
 \hc{最も}{motto-mo}
 \hc{最後}{sai-go}
 \hc{働く}{hatara-ku}
 \hc{海}{umi}}
```

書く
ka-ku
最も
motto-mo
最後
sai-go
働く
hatara-ku
海
umi

```
\font\label="Gentium Italic:color=006600" at 10pt
\def\listbox#1#2{\vtop{\hsize=60pt
 \hbox{\label #1}
 \moveright10pt\vbox{#2}}}
\font\t="Thonburi:color=660000" at 11pt
\font\k="AppleMyungjo Regular:color=660000" at 13pt
\hbox{\listbox{Thai}
  {\t กรกฎาคม\\กรง\\กว่า\\โกรธ\\ขรม\\ขวา\\โขยง\\ควัม}
 \listbox{Korean}
  {\k 게\\금\\길\\깅\\깥\\낙\\남}}
```

Thai
กรกฎาคม
กรง
กว่า
โกรธ
ขรม
ขวา
โขยง
ควัม

Korean
게
금
길
깅
깥
낙
남

Figure 3: Further fragments from the collation presentation, showing Asian scripts.

```
\font\arfont="Scheherazade:script=arab" at 13pt
\def\ar#1{{\arfont \beginR #1\endR}}
Burki, Rozi Khan. 1999. {\it
 \ar{ماخ ا خوئ زبان ته گورغاړه زر ژيين؟}
  [Should we leave our language on its
   death-bed?]}. Tank, Pakistan:
  Muslim Book Agency.

Karimi, Abdul Hamid Khan. 1995. {\it
 \ar{اردو كوهستانی بول چال اور كوهستانی شعر و ادب}
 [Urdu-Kohistani conversation and Kohistani
 culture]}.  Bahrain, Swat, Pakistan:
 Kohistan Adab Academy.
```

Burki, Rozi Khan. 1999. ماخ ا خوئ زبان ته گورغاړه زر ژيين؟ *[Should we leave our language on its death-bed?]*. Tank, Pakistan: Muslim Book Agency.

Karimi, Abdul Hamid Khan. 1995. اردو كوهستانی بول چال اور كوهستانی شعر و ادب *[Urdu-Kohistani conversation and Kohistani culture]*. Bahrain, Swat, Pakistan: Kohistan Adab Academy.

Figure 4: From the bibliography of a paper that cites some Arabic-script references. Note the unshaped Arabic characters in the source text shown here; how this actually appears to a user will depend on the capabilities of the text editor used.

```
\c 1
\s Mawu ƒe nya to Mawu Vi la dzi va
\p
\v 1 Le blema la, Mawu ƒo nu na mía tɔgbuiwo zi
 geɖe to eƒe gbeƒãɖelawo dzi le mɔ vovovo nu.
\v 2 Ke le egbe ŋkeke mamlɛ siawo me la, Mawu ƒo nu
 na mí to Via dzi. Vi siae eya ŋutɔ tia be wòanyi
 nuwo katã dome. Eyama ke dzie Mawu to wɔ xexeame hã.
\v 3 Vi lae ɖe alesi Mawu ƒe ŋutikɔkɔe le la fia,
 eye Mawu ƒe nɔnɔme tututue le eya hã si.
 Vi lae tsɔ eƒe nya ƒe ŋusẽ la lé xexeme blibo
 la ɖe te. Esi eyama klɔ míaƒe nuvɔ̃wo ɖa vɔ la,
 eyi ɖabɔbɔ nɔ anyi ɖe Mawu Bubutɔgã si le dziƒo
 uĩ la ƒe nuɖusime.

\c 1
\s دنيا جي پيدائش
\p
\v 1 شروعات ۾ خدا زمين ۽ آسمان کي پيدا ڪيو.
\v 2 ان وقت زمين بيترتيب ۽ ويران هئي. اونهي سمنڊ
جو مٿاڇرو اوندهه سان ڍڪيل هو ۽ پاڻيءَ جي مٿان خدا
جي روح ڦيرا پئي ڪي
\v 3 تڏهن خدا حڪم ڏنو ته ”روشني ٿئي.“ سو روشني ٿي پيئي.
```

Mawu fe nya to Mawu Vi la dzi va

1 [1]Le blema la, Mawu ƒo nu na mía tɔgbuiwo zi geɖe to eƒe gbeƒãɖelawo dzi le mɔ vovovo nu. [2]Ke le egbe ŋkeke mamlɛ siawo me la, Mawu ƒo nu na mí to Via dzi. Vi siae eya ŋutɔ tia be wòanyi nuwo katã dome. Eyama ke dzie Mawu to wɔ xexeame hã. [3]Vi lae ɖe alesi Mawu ƒe ŋutikɔkɔe le la fia, eye Mawu ƒe nɔnɔme tututue le eya hã si. Vi lae tsɔ eƒe nya ƒe ŋusẽ la lé xexeme blibo la ɖe te. Esi eyama klɔ míaƒe nuvɔ̃wo ɖa vɔ la, eyi ɖabɔbɔ nɔ anyi ɖe Mawu Bubutɔgã si le dziƒo uĩ la ƒe nuɖusime.

دنيا جي پيدائش

۱ [۱]شروعات ۾ خدا زمين ۽ آسمان کي پيدا ڪيو. [۲]ان وقت زمين بيترتيب ۽ ويران هئي. اونهي سمنڊ جو مٿاڇرو اوندهه سان ڍڪيل هو ۽ پاڻيءَ جي مٿان خدا جي روح ڦيرا پئي ڪي [۳]تڏهن خدا حڪم ڏنو ته ”روشني ٿئي.“ سو روشني ٿي پيئي.

Figure 5: Using a TEX macro package designed for Scripture formatting, showing examples in both African (extended Latin script) and Pakistani (Arabic script) languages.

3 Extending the character set

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 of Unicode, 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 expects to use in working with Unicode text require UTF-16, so working with this encoding form avoids the need for conversion at this interface.
- There are a number of internal tables in the TEX program that are implemented as arrays indexed by character code. In standard TEX, these arrays have 256 elements each. Enlarging them to 65,536 elements each, to index them by UTF-16 code values, is just about reasonable; enlarging them further, to allow direct indexing by Unicode scalar values, would make for extremely large arrays. To keep the memory footprint reasonable (both at runtime and for "dumped" macro collections), some kind of sparse array implementation would probably be needed, requiring significant additional development and testing, and perhaps impacting performance of key inner-loop parts of the TEX system.
- These per-character arrays are used to implement character "categories", used in parsing input text into tokens, as well as case conversions and "space factor" (a property used to modify word spacing for punctuation in Roman typography). In practice, it seems unlikely that there will be a great need to customize these character properties for individual supplementary-plane characters. They're unlikely to be wanted as escape characters or other special categories of TEX input; need not have the "letter" property that allows them to be part of TEX control sequences; and probably don't need to be included in automatic hyphenation patterns.

In view of these factors, XETEX works with UTF-16 code units, and Unicode characters beyond U+FFFF cannot be given individually-customized TEX properties. They can still be included in documents, however, and will render correctly (given appropriate fonts) as the UTF-16 surrogate pairs will be properly 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.

4 Implementing a character/glyph model

An important aspect of rendering Unicode text is the character/glyph model; it is assumed that the reader is familiar with this concept. Traditionally, TEX does not have a well-developed character/glyph

model. Input text is a sequence of 8-bit codes, interpreted as character tokens or other (e.g., control sequence) tokens according to the scanning rules and character categories. These same 8-bit codes are used as access codes for glyphs in fonts. It is possible to remap codes by TEX macro programming, and the "font metrics" (`.tfm`) files used by TEX can include simple ligature rules (e.g., `fi` ↦ fi), but the model is fairly rudimentary, and not adequate for script behaviors such as Arabic cursive shaping or Indic reordering. To support the full range of complex scripts in Unicode, a more complete character/glyph model is needed.

Rather than designing a text rendering system based on the Unicode character/glyph model from scratch, it seemed desirable to leverage existing implementations, allowing TEX to take advantage of the "smart fonts" and multilingual text rendering facilities found in modern operating systems and libraries. At the time of writing, XETEX supports two such rendering systems; it is possible that additional ones will be supported in future versions.

4.1 Using ATSUI on Mac OS X

The first smart-font rendering system implemented in XETEX was the ATSUI[1] system under Mac OS X. ATSUI is the Mac OS X component that renders Unicode text using AAT[2] fonts. The essential objects needed to render text with ATSUI are *text layouts* and associated *styles* (which in turn refer to *fonts* and other attributes).

In order for a system like ATSUI to render text correctly, it must be given complete runs of text, not individual characters; otherwise, behavior such as reordering and contextual glyph selection cannot happen. TEX normally treats each character of text as an individual node in a list, with known (and fixed) dimensions. Paragraph layout consists of taking a list of such nodes, with intervening "glue" (potentially flexible space) and other items, and determining the best sequence of line-break positions and the final location of each character and other node.

When using ATSUI for Unicode text, however, XETEX cannot treat each character (or, strictly speaking, UTF-16 code unit) as a separate node, to be measured and positioned individually. Instead, it collects sequences of characters that share the same font style, and calls ATSUI to measure such sequences (typically, entire words). A paragraph then consists of a list of such "word nodes", each with its dimensions as determined by ATSUI, and intervening space and other nodes. The basic TEX paragraphing algorithm applies just as well to these larger "chunks" as to traditional character nodes.

During formatting, then, XETEX makes use of just a few basic ATSUI APIs, in order to measure each word (or similar fragment) of text; in particular:

ATSUCreateStyle, ATSUSetAttributes Create an ATSUI style object, and assign appropriate text attributes. One ATSUStyle is associated with each font face and size combination requested by the TEX document, and used whenever text in that particular style needs to be measured.

ATSUCreateTextLayout, ATSUSetTextPointerLocation, ATSUSetRunStyle Create an ATSUI text layout object, and associate a string of Unicode text and a style object with it.

ATSUGetUnjustifiedBounds Measure a range of text as rendered with the associated font and other attributes. This gives the TEX paragraphing algorithm the measurements that it will use in laying out the text.

[1] Apple Type Services for Unicode Imaging; see `http://developer.apple.com/intl/atsui.html`.

[2] Apple Advanced Typography; see `http://developer.apple.com/fonts/TTRefMan/RM06/Chap6AATIntro.html`.

(In addition, a number of font-related ATSUI APIs are used to enumerate the fonts available in the system, determine what layout features the fonts support, etc.)

When XETEX has completed layout for a paragraph of text, therefore, it has a list of lines each containing a list of "word nodes"; each such node contains a run of Unicode text and a reference to an ATSUI style. The TEX system *does not know* the details of the actual glyphs that will be used to render the text, or precisely where they will be positioned; only the overall dimensions and position of each word. The glyph-level detail is left entirely to the ATSUI rendering system.

To illustrate this, figure 6 shows the trace output generated for a short fragment of text set as a single-line paragraph. Using a standard TEX font, the line consists of a list of character nodes, with intervening "glue" (flexible space). Each character has an associated font, and its metrics will be looked up in the corresponding TEX font metric (TFM) file. Note that TEX has automatically inserted some kerns between adjacent letters; this is also controlled by the TFM file for this font. In contrast, using an AAT font, the text line consists simply of a list of complete words with intervening glue; TEX does not deal with the individual characters. There may be kerning between letters here too, but the TEX algorithms are unaware of it; it happens automatically as ATSUI measures or renders the words, and TEX knows only their overall dimensions.

```
\font\AATfont="Times Roman" \AATfont
\setbox0=\vbox{The quick brown fox.}
\showbox0

\font\TeXfont=cmr10 \TeXfont
\setbox1=\vbox{The quick brown fox.}
\showbox1

> \box0=
\vbox(8.0+2.0)x469.75499
.\hbox(8.0+2.0)x469.75499, glue set 363.10948fil
..\hbox(0.0+0.0)x20.0
..\AATfont The
..\glue 2.5 plus 1.66666 minus 0.83333
..\AATfont quick
..\glue 2.5 plus 1.66666 minus 0.83333
..\AATfont brown
..\glue 2.5 plus 1.66666 minus 0.83333
..\AATfont fox.
..\penalty 10000
..\glue(\parfillskip) 0.0 plus 1.0fil
..\glue(\rightskip) 0.0
```

```
> \box1=
\vbox(6.94444+1.94444)x469.75499
.\hbox(6.94444+1.94444)x469.75499, glue set 356.6715fil
..\hbox(0.0+0.0)x20.0
..\TeXfont T
..\TeXfont h
..\TeXfont e
..\glue 3.33333 plus 1.66666 minus 1.11111
..\TeXfont q
..\TeXfont u
..\TeXfont i
..\TeXfont c
..\kern-0.27779
..\TeXfont k
..\glue 3.33333 plus 1.66666 minus 1.11111
..\TeXfont b
..\TeXfont r
..\TeXfont o
..\kern-0.27779
..\TeXfont w
..\TeXfont n
..\glue 3.33333 plus 1.66666 minus 1.11111
..\TeXfont f
..\TeXfont o
..\kern-0.27779
..\TeXfont x
..\TeXfont .
..\penalty 10000
..\glue(\parfillskip) 0.0 plus 1.0fil
..\glue(\rightskip) 0.0
```

Figure 6: Using tracing commands to examine XETEX's internal structures for text using an AAT font, compared to the same text with a standard TEX font.

After document formatting is complete, the XETEX "back-end" (actually a separate process, `xdv2pdf`) reads the `.xdv` file that encodes the formatted document, and creates a PDF version for viewing or printing. To do this, it "renders" the page encoded in the `.xdv` file through the Mac OS X Quartz graphics system, with a PDF file as the rendering destination. At this point, it again uses ATSUI APIs, loading each text string into an ATSUTextLayout, assigning the proper style, and calling **ATSUDrawText** to image the text into the PDF being constructed.

4.2 Using OpenType via ICU Layout

While the initial implementation of XETEX was based on Apple's ATSUI rendering system, the increasing availability of fonts with OpenType layout features led to a desire to also support this font technology. The system was therefore extended by incorporating the OpenType layout engine from ICU[3]. (In addition to the actual layout engine, XETEX makes use of ICU's implementation of the Unicode Bidi Algorithm.) The main functions used in the typesetting process include:

ubidi_open, ubidi_close, ubidi_setPara, ubidi_getDirection, ubidi_countRuns, ubidi_getVisualRun
: Before laying out glyphs, it is necessary to deal with bidirectional layout issues; most "chunks" XETEX needs to measure will be unidirectional, but this is not always the case. With mixed-direction text, each direction run is measured separately.

LayoutEngine::layoutChars, getGlyphs, getGlyphPositions
: The ICU LayoutEngine class is used to perform the actual layout process, and retrieve the list of glyphs and positions. The resulting array of positioned glyphs is stored within the "word node" in XETEX's paragraph list.

Internally, ICU-based OpenType rendering is handled in a very different way from ATSUI rendering. With ATSUI, the output of the typesetting process includes the original Unicode strings and the appropriate font descriptors; the PDF-generating back-end then reuses ATSUI layout functions to actually render the text into the PDF destination. In the case of OpenType, however, the typesetting process retrieves the array of positioned glyphs that result from the layout operation, and records this; the back-end then merely has to draw the glyphs as specified, not repeat any of the text layout work.

When the TEX source calls for a particular font, XETEX looks for specific layout tables within the font (e.g., `morx` for AAT, or `GSUB` for OpenType) to determine which layout engine to use, and instantiates either an ATSUI style or an ICU LayoutEngine as appropriate. The difference in the implementation of the two technologies is, however, entirely hidden from the main TEX program, which simply deals with "word nodes", forming them into paragraphs and pages once they've been measured by the appropriate smart-font engine.

4.3 Hyphenation support

Implementing "word nodes" as "black boxes" within the main TEX program made it easy to form paragraphs of such words, without extensive changes to the rest of TEX. A complication arose, however, in that TEX has an automatic hyphenation algorithm that comes into effect if it is unable to find satisfactory line-break positions for a paragraph. The hyphenation routine applies to lists of character nodes representing runs of text within a paragraph to be line-broken. But at this level, the program sees Unicode "word nodes" as indivisible, rigid chunks.

[3] IBM's open-source project, International Components for Unicode; see `http://oss.software.ibm.com/icu/`.

Explicit discretionary hyphens may 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 between them.

In order to reinstate hyphenation support, therefore, 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 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 hyphen nodes in between.

A final refinement proved necessary here: once the line-breaks have been chosen, and the lines of text are being "packaged" for 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 hyphenation points. With an early release of XETEX, a user reported that OpenType ligatures in certain words such as *different* would intermittently fail (appearing as *different* instead), and this turned out to be caused when automatic hyphenation came into effect and a discretionary break was inserted.

5 Backward compatibility

The original motivation for the XETEX project was to provide a typesetting solution that worked with Unicode and complex scripts, via smart font technologies. However, it soon became clear that many existing TEX users, with no complex-script requirements, nevertheless found the integration with the host platform's font management to be very attractive, and wished to use XETEX and native Mac OS X fonts with existing TEX (or more commonly LATEX) documents. There is a huge legacy of pre-Unicode TEX documents and resources, and it is helpful for users to be able to continue working with these materials, while at the same time beginning to take advantage of the extended capabilities of XETEX.

5.1 Traditional TEX input conventions

Existing TEX and LATEX documents often use ASCII-based sequences to represent accented and other "extended" characters not directly available in the input character set. The macro packages that implement these commands map them to known character codes in particular font encodings.

To allow such documents to be typeset using standard Unicode-compliant fonts in place of the custom-encoded fonts previously used, Ross Moore (an early XETEX user) has provided a package[4] for LATEX that maps several hundred such control sequences to the correct Unicode codepoints. Using this package, many existing LATEX documents that use extended Latin and other "special" characters can be typeset using Unicode fonts, without needing to actually convert the encoding of the source text.

5.2 Legacy source document encodings

As initially designed, XETEX assumed that all input text is encoded in Unicode; it would read input files as either UTF-8 or UTF-16. Existing ASCII documents, of course, are also valid UTF-8 and therefore could be used directly. This includes documents that use ASCII-based TEX conventions for accents and other extended characters, as mentioned above.

[4] See the *utf8accents* package, available from `http://scripts.sil.org/xetex_related`.

However, some TEX users have documents encoded with 8-bit codepages such as ISO Latin-1, MacRoman, Windows Cyrillic, etc. With the original "pure Unicode" implementation of XETEX, it was impossible to process such files; they would be assumed to be UTF-8, but on encountering values > 127, the bytes would be misinterpreted as UTF-8 sequences rather than as individual characters. (In standard TEX, with purely byte-oriented input, such files can of course be read; and the characters can be remapped through TEX macro programming, if (as often occurs) there is a mismatch between the encodings of input text and the fonts to be used.

To enable users to process such files with XETEX, without requiring a separate conversion to Unicode first, more flexible input encoding support was eventually added to the system. A new command `\XeTeXinputencoding` was implemented, which allows the user to request on-the-fly conversion from another character encoding into Unicode as the source text is read.

The `\XeTeXinputencoding` command requires one parameter, the name of the desired encoding. A number of options are supported. First, `utf8` or `utf16` will set the system to direct Unicode input (or `auto` restores the default behavior, which is to detect the Unicode encoding form from the initial bytes of the file). The special name `bytes` causes XETEX to read individual byte values as separate code units, treating them as character codes 0–255. While this is unlikely to represent the correct Unicode interpretation of the source text, it may be useful if these codes are to be processed by existing TEX macros rather than used directly as text characters.

Finally, any Internet encoding name known to the Mac OS X Text Encoding Converter[5] may be specified. In this case, XETEX calls TEC to perform encoding conversion as it reads the input text. Just a few basic TEC APIs are sufficient for this task:

TECGetTextEncodingFromInternetName Used by `\XeTeXinputencoding` to look up the encoding name specified, and determine if it is known to the operating system.

CreateTextToUnicodeInfo Initialize the mapping information needed by TEC to convert between a particular legacy encoding and Unicode.

ConvertFromTextToUnicode Convert a buffer of text from the external legacy encoding into Unicode.

Note that although at the time of writing, XETEX relies on TEC for encoding conversion of input text, this may change in a future release. A future version will probably use either ICU or GNU *libiconv* functions instead of TEC. This would be in the interests of portability to operating systems other than Mac OS X.

It is also possible that the input mapping support will be extended to allow the use of SIL's TECkit[6] to directly support custom user-defined byte encodings. This would involve a minor extension to the `\XeTeXinputencoding` command, allowing users to specify the name of a TECkit mapping file as an alternative to the name of a standard legacy codepage.

5.3 Font mappings using TECkit

Widely-used TEX keyboarding conventions such as `\'{e}` ↦ 'é' or `\pounds` ↦ '£' are implemented via TEX macros (and therefore easily adapted for Unicode-compliant fonts, by modifying the macro definitions). In addition, there are a few established conventions that are implemented as ligature rules

[5] A standard component of the Mac OS; see `http://developer.apple.com/documentation/Carbon/Reference/Text_Encodin_sion_Manager/index.html`.

[6] Text Encoding Conversion toolkit; see `http://scripts.sil.org/teckit`.

```
; TECkit mapping for TeX input conventions <-> Unicode characters
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 double quote
U+0022                > U+201D ; " -> right double quote

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

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

Figure 7: The TECkit source file `tex-text.map`, defining a font mapping for XETEX that provides compatibility with the conventions of legacy TEX fonts.

associated with standard TEX fonts; these include `---` ↦ '—' (em-dash), `?`` ↦ '¿', and a few more. In principle, smart font technologies such as AAT and OpenType could implement these same ligatures, providing the same behavior as traditional TEX fonts. But as these conventions are peculiar to the TEX world, it is not realistic to expect them to be provided in mainstream, general-purpose fonts.

Although it would usually be possible to simulate these ligatures via macro programming, it is difficult to ensure that reprogramming widely-used text characters such as the hyphen, question mark, and quotation marks will not interfere with other levels of markup in the source document. Instead, XETEX provides a mechanism known as "font mappings", whereby a mapping of Unicode characters is associated with a particular font, and applied to all strings of text being measured or rendered in that font. This is implemented using the TECkit mapping engine.

Although primarily designed to convert between legacy byte encodings and Unicode, TECkit can also be used to perform transformations on a Unicode text stream, using the same mapping language; figure 7 shows the *tex-text* mapping that is provided to support normal TEX conventions. When associated with a standard Unicode-compliant font in XETEX, this has the effect of implementing the legacy TEX conventions for dashes and quotes, as shown in figure 8, without requiring any TEX-specific features in the smart fonts themselves.

While this mechanism, associating a mapping defined in terms of Unicode character sequences, was first devised in order to support legacy TEX input conventions, it can also be applied in other ways. For example, figure 9 shows how it is possible to typeset a single fragment of input text in two scripts by giving different font specifications, one of which includes a transliteration mapping.

5.4 Math typesetting

One of TEX's traditional strengths is in mathematical typesetting. It was designed to enable authors to readily typeset complex equations and similar displays, with precise control over details of layout and

```
\font\TestA="Times New Roman" at 9pt
\TestA !‘Typing ‘‘quotes’’---and
  dashes---the \TeX\ way!\par
\bigskip
\font\TestB="Times New Roman:
  mapping=tex-text" at 9pt
\TestB !‘Typing ‘‘quotes’’---and
  dashes---the \TeX\ way!\par
```

!`Typing ``quotes"---and dashes---the TEX way!

¡Typing “quotes”—and dashes—the TEX way!

Figure 8: Using the *tex-text* font mapping to support legacy typing conventions.

```
\begin{centering}
\def\SampleText{Unicode - это уникальный
   код для любого символа,\\
    независимо от платформы,\\
    независимо от программы,\\
    независимо от языка.\par}
\font\gen="Gentium" at 9pt \gen
\SampleText
\bigskip
\font\gentrans="Gentium:
  mapping=cyr-lat-iso9" at 9pt \gentrans
\SampleText
\end{centering}
```

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

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

Figure 9: Using a font mapping for on-the-fly transliteration while typesetting.

spacing, and with many aspects of math formatting handled automatically. Figure 10 shows an example of TEX input using math mode, alongside the typeset result.

Standard TEX macro packages provide commands to access many hundreds of math and other technical symbols, and these are mapped to the appropriate characters in a selection of specialized fonts. In Unicode many such symbols now have their own codepoints, and so it should be possible to typeset such material using standard Unicode-compliant fonts, rather than the custom-encoded math

```
$$\eqalign{
\biggl(\int_{-\infty}^\infty e^{-x^2}\,dx\biggr)^2
&=\int_{-\infty}^\infty\int_{-\infty}^\infty
  e^{-(x^2+y^2)}\,dx\,dy\cr
&=\int_0^{2\pi}\int_0^\infty e^{-r^2}r\,dr\,d\theta\cr
&=\int_0^{2\pi}\biggl(-{e^{-r^2}\over2}
  \bigg\vert_{r=0}^{r=\infty}\,\biggr)\,d\theta\cr
&=\pi.\cr}$$
```

$$\begin{aligned}\left(\int_{-\infty}^{\infty} e^{-x^2}\,dx\right)^2 &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-(x^2+y^2)}\,dx\,dy\\ &= \int_0^{2\pi}\int_0^{\infty} e^{-r^2} r\,dr\,d\theta\\ &= \int_0^{2\pi}\left(-\frac{e^{-r^2}}{2}\Bigg|_{r=0}^{r=\infty}\right)d\theta\\ &= \pi.\end{aligned}$$

Figure 10: An example of math typesetting, one of TEX's strengths.

and symbol fonts normally used with TEX.

However, this is not as simple as it may sound. TEX's math typesetting features rely heavily on specialized font metric information associated with the math fonts, in addition to the glyphs themselves. This is necessary in order to provide accurate typesetting of complex constructs. Unfortunately, this means that users cannot simply tell XETEX to use a Unicode-compliant font for math; not only will the standard TEX math commands access the wrong glyphs, but also, even if the TEX macros were redefined to access the proper Unicode values, such fonts will still not work in math mode because of the lack of extended font metric information.

To address this issue, it will be necessary to provide these additional font metrics for any Unicode fonts that are to be used for math typesetting; and it will also be necessary to extend additional data structures in XETEX that are currently limited to 8-bit codes. It may be possible to make use of work from the Omega project (see section 7.3) to facilitate this, but at the time of writing, XETEX is limited to using legacy 8-bit TEX fonts for math typesetting.

6 Advanced font features

In order to take full advantage of the multilingual support and sophisticated typographic features offered by modern font technologies, we need to go beyond merely specifying a font and rendering a sequence of Unicode characters. Correct rendering may depend on the language of the text, as different languages sometimes require different visual results even for the exact same Unicode characters.

For example, Turkish uses both of the characters *i* (U+0069 LATIN SMALL LETTER I) and *ı* (U+0131 LATIN SMALL LETTER DOTLESS I). But many Latin-script fonts include an *fi* ligature in which the dot of the *i* is assimilated into the top of the *f*. Such a ligature is appropriate for most languages, and improves the appearance of the rendered text; and typical OpenType or AAT fonts will contain ligature rules that automatically use it wherever the sequence `fi` occurs in the text stream. However, in Turkish it becomes a problem because the distinction between *i* and *ı* is lost. Figure 11 illustrates this issue.[7]

```
\font\txtfont="Adobe Garamond Pro" at 8pt
\txtfont
Apple, HP, IBM, JustSystem, Microsoft, Oracle,
SAP, Sun, Sybase, Unisys ve endüstrinin diğer
ileri gelen {\red firmaları} Evrensel Kod
Standardını desteklemektedirler. Evrensel Kod,
XML, Java, ECMAScript (JavaScript), LDAP, CORBA
3.0, WML vb. gibi modern standartlar
{\red tarafından} ISO/IEC 10646 uyarlanmasının
resmi yoludur.
```

Apple, HP, IBM, JustSystem, Microsoft, Oracle, SAP, Sun, Sybase, Unisys ve endüstrinin diğer ileri gelen firmaları Evrensel Kod Standardını desteklemektedirler. Evrensel Kod, XML, Java, ECMAScript (JavaScript), LDAP, CORBA 3.0, WML vb. gibi modern standartlar tarafından ISO/IEC 10646 uyarlanmasının resmi yoludur.

Figure 11: Turkish text showing the necessity for language-specific font rendering. Note how the automatic use of the *fi* ligature obscures the distinction between *i* and *ı*, which are separate letters in Turkish.

An increasing number of OpenType fonts provide a solution to this problem, in the form of support for multiple "language systems". The font developer can provide tables within the font that substitute different glyphs, enable different subsets of the possible ligatures, etc., depending on the selected language. XETEX supports this capability by allowing a font specification in the TEX document

[7] Sample Turkish text from the Unicode web site, `http://www.unicode.org/standard/translations/turkish.html`.

```
\font\txtfont="Adobe Garamond Pro:
  language=TUR" at 8pt
\txtfont
Apple, HP, IBM, JustSystem, Microsoft, Oracle,
SAP, Sun, Sybase, Unisys ve endüstrinin diğer
ileri gelen {\red firmaları} Evrensel Kod
Standardını desteklemektedirler. Evrensel Kod,
XML, Java, ECMAScript (JavaScript), LDAP, CORBA
3.0, WML vb. gibi modern standartlar
{\red tarafından} ISO/IEC 10646 uyarlanmasının
resmi yoludur.
```

Apple, HP, IBM, JustSystem, Microsoft, Oracle, SAP, Sun, Sybase, Unisys ve endüstrinin diğer ileri gelen firmaları Evrensel Kod Standardını desteklemektedirler. Evrensel Kod, XML, Java, ECMAScript (JavaScript), LDAP, CORBA 3.0, WML vb. gibi modern standartlar tarafından ISO/IEC 10646 uyarlanmasının resmi yoludur.

Figure 12: Using the Turkish language support in an OpenType font to ensure correct rendering of `fi` sequences.

to include a language tag. If the specified language tag is supported by the font, XETEX will render the font according to those OpenType rules instead of the default behavior. Figure 12 shows how the Turkish example can be corrected by adding the proper language tag to the font declaration.

Another issue is that some Unicode characters have alternative possible glyph shapes (within a single typeface design). For example, the character U+014A LATIN CAPITAL LETTER ENG has four possible designs in the Doulos SIL typeface: Ŋ Ɲ Ɲ Ɲ. These are all legitimate renderings of the same Unicode character, the uppercase version of *eng* (ŋ). By default, text rendered in Doulos SIL will use the first of these forms. But in some language communities, a different form may be preferred—or may even be required for readability.

One solution to this would be to use different language systems within the OpenType tables to access the alternate *Eng* glyphs. In practice, however, this is more difficult to arrange than the Turkish case. This character is used in many lesser-known and little-documented languages, and so font developers cannot reasonably be expected to provide all the appropriate language system mappings.

An alternative approach is through user-selectable *font features*. These are another mechanism for controlling exactly how a given font renders text. Rather than associating variant glyphs or other options with a particular language, the options can be made available as individual "features" that can be enabled or disabled as required. So the Doulos SIL font, for example, has a feature named *Uppercase Eng alternates*, with four possible settings, and the character U+014A can be displayed as any of the four available glyphs according to the feature setting chosen.

XETEX allows such features to be included as part of the font specification used to load a particular font for typesetting. Figure 13 shows how the same text can be typeset either with the default glyphs provided in Doulos SIL, or using alternate forms for particular characters. This mechanism enables the user to achieve culturally-appropriate rendering of the Unicode text without requiring the font developer to be aware of the proper choices of glyph variants for each language where the font may be used.

7 XETEX and other TEX extensions

The XETEX project is just one of a number of extended versions of TEX that have been created over the years, and it may be useful to give a brief comment on its relationship to some other projects.

```
\font\Doulos="Doulos SIL/AAT" at 10pt
\font\DoulosAlt="Doulos SIL/AAT:
  Alternate forms=Literacy alternates,
    Small v-hook straight style;
  Uppercase Eng alternates=Capital N with tail"
    at 10pt
\Doulos
Xɔsee na Mose ɖo Ŋutitotoŋkeke la anyi, eye wòna
wohlẽ ʋu ɖe ʋɔtrutiwo ŋu bene dɔla si atsrɔ̃
ŋgɔgbeviwo la nagawɔ nuvevi Israel viwo ya o.\par
\bigskip
\DoulosAlt
Xɔsee na Mose ɖo Ŋutitotoŋkeke la anyi, eye wòna
wohlẽ ʋu ɖe ʋɔtrutiwo ŋu bene dɔla si atsrɔ̃
ŋgɔgbeviwo la nagawɔ nuvevi Israel viwo ya o.\par
```

Xɔsee na Mose ɖo Ŋutitotoŋkeke la anyi, eye wòna wohlẽ ʋu ɖe ʋɔtrutiwo ŋu bene dɔla si atsrɔ̃ ŋgɔgbeviwo la nagawɔ nuvevi Israel viwo ya o.

Xɔsee nɑ Mose ɖo Ŋutitotoŋkeke lɑ ɑnyi, eye wònɑ wohlẽ ʋu ɖe ʋɔtrutiwo ŋu bene dɔlɑ si ɑtsrɔ̃ ŋgɔgbeviwo lɑ nɑgɑwɔ nuvevi Isrɑel viwo yɑ o.

Figure 13: Using optional font features to control the rendering of specific Unicode characters. This may be for stylistic reasons, or may be required for readability in a particular language community.

7.1 TEXGX

TEXGX[8] is an important ancestor of XEHTEX, in that it pioneered the model of integrating the TEX formatting system with a host platform's smart-font rendering technology. TEXGX relied on Apple's now-obsolete QuickDraw GX graphics system for Mac OS 7–9, and was still based on 8-bit legacy encodings, not Unicode. However, it did adapt TEX's scanning and paragraphing routines to treat entire words as units to be passed to an external text layout system, as well as extending the `\font` command to load fonts from the host platform and to access optional features. These extensions have been incorporated largely unchanged into XEHTEX.

7.2 ε-TEX

ε-TEX[9] is a widely-used extended version of TEX that adds a number of new commands to the language, while retaining compatibility with the standard program. It is in fact used by many installations as the default 'TEX' program, and an increasing number of macro packages assume that ε-TEX features are available.

XEHTEX is implemented as an extension of ε-TEX, so that it benefits from the enhancements provided in that system. In particular, support for bidirectional paragraph layout, needed for languages such as Arabic and Hebrew, is inherited from ε-TEX.

7.3 Omega, Aleph

Omega[10] is an ambitious project that extends TEX to work with 16-bit character codes and provides a mechanism of input and output filters ("Omega transformation processes"). These can perform complex

[8] See `http://www.sil.org/computing/texgx.html`.

[9] See `http://www.tug.org/tex-archive/systems/e-tex/v2/`.

[10] See `http://omega.enstb.org/` and `http://www.tex.ac.uk/cgi-bin/texfaq2html?label=omega`.

transformations both as text is read from a file (to support different encodings) and between the internal character codes and font access codes.

Aleph[11], formerly known as e-Omega, is a project that aims to merge features of Omega and ε-TEX, and to provide a more stable platform than Omega, which has been undergoing major restructuring and appears to face a somewhat uncertain future.

Omega (and therefore Aleph) provides a very powerful mechanism for supporting multilingual typesetting and complex scripts. However, because (like TEX itself) it is intended to be platform-independent, it does not take advantage of available text systems such as ATSUI; instead, all complex script behavior must be implemented through Omega's own OTP mechanisms. This is both a strength and a weakness: a strength in that the mechanism is both powerful and portable; but a weakness in that configuring fonts (especially for complex scripts) to work with Omega is a non-trivial programming task, beyond the capabilities of many users.

In contrast, one of the key ideas of XETEX is that it should *not* be necessary to re-implement complex script and typographic behavior that has already been defined by the developers of AAT and OpenType fonts. If a user's computer system supports a given writing system, with appropriate fonts and rendering behavior, this should be immediately usable in the typesetting system; no laborious, technical configuration procedure should be needed. This gives XETEX a major ease-of-use advantage, but comes at a price: the typesetting system is now dependent on the host platform's font technology, in a way that standard TEX is not, and so documents may not be readily portable to other platforms.

While XETEX does not share Omega's complex-script features, taking an entirely different approach to text rendering, it does use ideas from Omega with regard to the "widening" of character codes within the TEX engine from 8 to 16 bits.

7.4 pdfTEX

One more TEX extension is worth mentioning here: pdfTEX is an extended version of TEX that provides the option to generate PDF output, instead of the traditional DVI that requires post-processing with a device-specific driver. It includes a number of additional commands to enhance the resulting PDF files with bookmarks, interactivity, and so on.

While XETEX also generates PDF output, this is actually accomplished by generating "extended DVI" output, and then rendering this to PDF as a subsequent process. This is less efficient than pdfTEX's direct PDF generation, and makes it more difficult to integrate some of the advanced PDF features (although to some extent, this can still be done through the traditional DVI-to-device driver model).

At present, then, XETEX is entirely separate from pdfTEX. Integration of XETEX's Unicode and font support with pdfTEX's PDF generation and extended PDF features would provide an even more attractive typesetting system, but would require significant development effort; at the time of writing, no such effort is under way.

8 Future directions

This paper has discussed how Unicode and multilingual/multi-script support has been integrated into the TEX system, providing users with a powerful typesetting system that handles virtually any language for which an appropriate "smart font" is available. However, XETEX should still be considered somewhat

[11] See `http://www.tex.ac.uk/cgi-bin/texfaq2html?label=aleph`.

experimental, a "work in progress", and there is considerable scope for further enhancement. A few possible directions for further work include:

Graphite support Another smart font technology, besides AAT and OpenType, is SIL International's Graphite[12] system. This could be integrated as a third text rendering option within XETEX.

Alternate platforms Currently, XETEX is available only on Mac OS X. It was initially developed on this platform by taking advantage of specific Mac OS X technologies (especially ATSUI for Unicode text layout, and Quartz2D for graphics/PDF rendering). However, there is considerable interest in porting to other operating systems.

pdfTEX integration As suggested above, it would seem ideal to merge XETEX's Unicode support with pdfTEX's PDF generation.

Unicode math font support This has also been mentioned above; currently, only legacy 8-bit fonts can be used for math typesetting. Fully supporting Unicode for math will probably require coordination among font developers and TEX macro writers, as well as extensions to the XETEX system itself.

Line-breaking without spaces Writing systems that do not use spaces between words are currently not well supported in XETEX. Line-breaking and paragraph layout relies on recognizing potential line break positions, primarily at spaces, and so will not work in languages such as Thai or Chinese. It is possible to work around this by use of zero-width spaces in the source text, but ideally the paragraphing algorithm should be extended to handle such writing systems correctly.

While these are offered as examples of how XETEX, or perhaps some future system modeled on the current project, might develop further, this does *not* constitute a commitment to implement any particular feature!

[12] See `http://scripts.sil.org/RenderingGraphite`.