

The X_YTeX Companion

TeX meets OpenType and Unicode

Edited by Michel Goossens (CERN)

Work in progress. Version January 11,2010

Please send your comments to michel.goossens@cern.ch

The copyright of the contributions extracted from documentation of the various packages (see below for details) remains with their respective authors. The current maintainer of this document is Michel Goossens.

Work history

- *January 2008* Initial version (from LGC2 supplementary material).
- *Spring 2008* Adapted material from Jonathan Kew's X_YTeX manual and Will Robertson's fontspec manual.
- *January 2009* Adapted material from François Charette's arabxetex manual and Dian Yin's zhspacing manual.
- *July 2009* Added material contributed by Vafa Khalighi describing his bidi package.
- *August 2009* Added material about xecjk plus introduced corrections and clarifications suggested by Leo Ferres and Karel Píška.
- *January 2010* Added lots of corrections and a few suggestions for clarifications by Taylor Venable.

Contents

List of Figures	vii
List of Tables	ix
Preface	xi
1 PostScript fonts and beyond	1
1.1 Font formats: a brief history	2
1.1.1 Adobe and its PostScript Type 1	2
1.1.2 TrueType fonts	3
1.1.3 Two competing technologies	3
1.1.4 The best of two worlds: OpenType	3
1.2 PostScript Type 1 and TrueType: two different approaches	4
1.2.1 Interoperability	4
1.3 Unicode: the universal character encoding	5
1.4 OpenType	5
1.4.1 OpenType tables	6
1.4.2 OpenType features	11
1.4.3 OpenType support today	12
1.4.4 Interrogating OpenType fonts	13
2 X _Y TeX: T _E X meets OpenType and Unicode	19
2.1 X _Y TeX: a historical introduction and some basics	21
2.1.1 A brief history	22
2.1.2 X _Y TeX: basic principles	22
2.2 X _Y TeX: typesetting with glyphs, characters and fonts	23
2.2.1 Accessing font with fontconfig	23
2.2.2 Specifying character codes	25
2.2.3 Hyphenation	26
2.2.4 Font management: the basics	27
2.2.5 Font mappings using TECKit	28

2.2.6	Line breaks and justification	29
2.2.7	Unicode Character/glyph model	30
2.2.8	Using OpenType via ICU Layout.	30
2.2.9	X ₃ TeX's hyphenation support	32
2.2.10	Running xetex	33
2.3	Supplementary commands introduced by X ₃ TeX.	33
2.3.1	Specifying languages and scripts	34
2.3.2	Specifying optional features	35
2.3.3	Support for pseudo-features	36
2.3.4	Commands extracting information from OpenType fonts	36
2.3.5	Maths fonts.	39
2.3.6	Encodings, linebreaking, etc.	41
2.3.7	Graphics and pdfTeX-related commands	42
2.4	fontspec	43
2.4.1	Usage	43
2.4.2	Latin Modern defaults	43
2.4.3	Maths 'fiddling'	43
2.4.4	A first overview	44
2.4.5	Font selection	44
2.4.6	Default font families.	45
2.5	X ₃ TeX and other engines	47
3	Handling all those scripts	49
3.1	Writing systems	49
3.1.1	Basic terminology	50
3.1.2	History of writing systems	50
3.1.3	Types of writing systems	51
3.1.4	Language Resources	54
3.1.5	Freely available Unicode encoded fonts	54
3.1.6	Directionality	54
3.1.7	Writing systems on computers	55
3.2	Bidirectional typesetting.	55
3.2.1	Using The bidi Package	55
3.2.2	Basic Direction Switching	56
3.2.3	Typesetting Short RTL and LTR texts.	57
3.2.4	Multicolumn Typesetting	57
3.2.5	More peculiarities for RTL typesetting	58
3.2.6	Tabular material in RTL mode.	60
3.3	Languages using the Arabic alphabet	61
3.3.1	ArabTeX: Arabic typography with TeX	62
3.3.2	ArabX ₃ TeX: Arabic typography with X ₃ TeX.	64
3.3.3	Arabic presentation forms	75
3.4	Typesetting Chinese	79
3.4.1	The xeCJK Package	79
3.4.2	The zhspacing package	84
3.5	Examples of the use of Unicode	88
3.5.1	Unicode fonts and editors	88
3.5.2	Examples of Unicode texts	89

4	Unicode mathematics	91
4.1	Unicode for handling math across platforms and applications	91
4.2	X _Y TeX handling mathematics fonts	92
	Index of Commands and Concepts	95
	People	100

List of Figures

1.1	Using OpenType's advanced typographic features in Adobe InDesign	13
1.2	Opentype Unicode support in OpenOffice	14
1.3	Microsoft's <i>Fonts Extension</i>	15
2.1	Complexities when dealing with various languages	19
2.2	Scripts used in various parts of the world	20
2.3	Asian scripts	20
2.4	List of features for the scripts and languages supported by the Microsoft Arial and Adobe Minion fonts	40
3.1	Writing systems used in the world today	51
3.2	Examples of six Arabic calligraphic styles	62

List of Tables

2.1	Mathematics symbol types	39
3.1	Indic consonant–vowel combinations in various Indic abugidas	54
3.2	Arab \TeX 's input conventions for Arabic and Persian	63
3.3	All arabxetex input conventions.	75

Preface

This free booklet describes X_YTeX and its X_YL^ATeX variant. After an introduction to the OpenType and Unicode technologies, it describes how X_YTeX extends the TeX engine to optimally use OpenType fonts directly and allow you to handle Unicode-encoded sources.

Various L^ATeX packages have been developed recently to take advantage of X_YTeX's new functionalities, and those are described next.

This compilation of tools has been written in close collaboration with the authors: Jonathan Kew (X_YTeX development), Will Robertson fontspec and unicode-math), François Charette (arabxetex), and Dian Yin (zhspacing). Corrections and feedback has also been received from Adam Buchbinder, Leo Ferres, Rik Kabel, and Karel Píška.

Comments are welcome and can be addressed to `michel.goossens@cern.ch`.

Michel Goossens
January 2010

PostScript fonts and beyond

1.1 Font formats: a brief history.	2
1.2 PostScript Type 1 and TrueType: two different approaches	4
1.3 Unicode: the universal character encoding.	5
1.4 OpenType	5

In this chapter we look at the most basic type of graphical object in documents: the characters that form the words. Character shapes (“glyphs”) are not a direct part of the $\text{T}_{\text{E}}\text{X}$ system; all $\text{T}_{\text{E}}\text{X}$ wants to know about them is some metric information, such as their width or height. It is the task of the post-processing stage (the backend of $\text{pdfT}_{\text{E}}\text{X}$ or a device driver, such as dvips which reads the .dvi file as output by $\text{T}_{\text{E}}\text{X}$) to produce the actual graphical representation of the page. For this stage information about the actual shapes of the characters is needed and this information is stored in so-called fonts (collections of characters) for which many different storage formats exist. Thus in principle any existing font can be used with $\text{T}_{\text{E}}\text{X}$ provided that the metric information $\text{T}_{\text{E}}\text{X}$ needs is available or can be generated and that a procedure exists that understands the format in which the fonts are stored and can insert it into the output file.

Donald Knuth developed a companion program to $\text{T}_{\text{E}}\text{X}$, MetaFont , for generating fonts to be used with $\text{T}_{\text{E}}\text{X}$ (Chapter 3 of *The LaTeX Graphics Companion* looked briefly at MetaFont ’s drawing capabilities). For quite some time only fonts designed with MetaFont were available to $\text{T}_{\text{E}}\text{X}$ users, with the result that $\text{T}_{\text{E}}\text{X}$ or $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ documents had an easily identified look and feel—mainly a result of the use of the Computer Modern fonts. Given that the $\text{T}_{\text{E}}\text{X}$ community is very small compared to that of other typesetting systems very few font designers have produced fonts in MetaFont . Therefore, access for $\text{T}_{\text{E}}\text{X}$ engines to the literally thousands of fonts available commercially in other formats, in particular PostScript, TrueType, and, more recently, OpenType, has become a *must*.

Although at the beginning it was quite difficult to integrate PostScript fonts into $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ packages, the release of $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X} 2_{\epsilon}$ and its new font selection scheme (NFSS, see Chapter 7 of [5]) made accessing the large set of PostScript fonts more straightforward. Nowadays, documents routinely combine $\text{T}_{\text{E}}\text{X}$ ’s superior typesetting quality with all the professionally designed typefaces produced, mainly in PostScript, but also in TrueType and OpenType. The current chapter will introduce you to solutions to achieve this in a convenient way.

After a historic overview of modern font technologies, including a brief description of their respective technical capabilities, we take a closer look at the basic issues concerned with typesetting and how $\text{T}_{\text{E}}\text{X}$ and PostScript, working together, address this problem (how metric information is handled, the different types of $\text{T}_{\text{E}}\text{X}$ and PostScript fonts, how they are encoded, i.e., how one can access individ-

ual characters of a font, etc.) We then explain how you can use the “basic” PostScript fonts, as they are defined in the PSNFSS system (a collection of small packages and accompanying files for \LaTeX), which makes it easy to use a large number of common PostScript fonts out of the box) and how to easily download and install a few instances of freely available fonts. We extend the discussion to where to download and install the \LaTeX support files for commercially available fonts that you might have bought. Since many \LaTeX users have de facto access to a lot of TrueType fonts that come with their operating system, we devote the next section to the use of TrueType fonts with \pdf , in particular how one can use a large Unicode TrueType font for typesetting in many different scripts and languages. We are then ready to discuss a few recent \LaTeX packages which take advantage of the enriched possibilities of the OpenType technology. We end the chapter with a discussion of *Fontname*, also known as the “Berry” font naming scheme, which is important to uniquely identify and handle all \LaTeX support files of the large number of fonts that are available on current operating systems.

1.1 Font formats: a brief history

The current main font formats are PostScript Type 1 (Type 1), TrueType (TT), and OpenType (OT), an integrated superset of the first two. All three are based on font outline technologies, are multi-platform, and have their technical specifications openly available. These formats can be run on any recent computer platform and their character outlines (“glyphs”) are described mathematically as functions operating on points, lines and curves. The character representations are resolution independent and can be scaled to any size. These technologies implement “hinting” by associating additional information with each character to help the rasterization engine optimize their representation on any given output device.

1.1.1 Adobe and its PostScript Type 1

When Adobe launched PostScript in 1984, it supported two different types of fonts formats: Type 1,¹ the more sophisticated one with support for hinting and data compression, and Type 3, a more general (almost all PostScript graphics operators are allowed) but less optimized variant. At first Adobe did not publish the specification of its PostScript Type 1 format (the Type 3 spec was public), which helped Adobe take a large part of the commercial typography market but upset the other font foundries.

Apple, which also was founded in the early nineteen eighties, adopted PostScript as page description language for its Apple LaserWriter printer in 1985. Soon also other high-end image setting machines adopted PostScript as their native language. At about the same time the introduction of affordable desktop publishing software, such as Pagemaker, Freehand, set off a revolution in page layout technology, and PostScript backends appeared for most graphics programs, thus adding to the potential market for professional PostScript Type 1 fonts. Because of its reliability, its wide selection of fonts available, its clever rasterizing engine and superior hinting mechanism, historically PostScript has been the preferred font format of professional designers, publishers and printshops.

Concurrently Adobe had developed an “interactive” version of PostScript, called *Display PostScript*, that ran (somewhat slowly) on personal computers to allow displaying PostScript data on-screen. Although some computer manufacturers agreed to take out (and pay) software licences, Apple and Microsoft were quite unwilling to pay the royalties requested by Adobe and, moreover, to hand control to Adobe over a vital part of their operating system.

In the first part of the 1990s Adobe also developed the PostScript Type 1 multiple master (MM) format as an extension of PostScript Type 1. Essentially, it allows two (or more) design variations to be encoded on a given design axis (such as weight, width, optical size). Afterwards, any in-between state

¹See http://partners.adobe.com/public/developer/en/font/T1_SPEC.PDF.

(*instance*) may be generated by the user as required.¹

1.1.2 TrueType fonts

The major system software vendors (Apple, Microsoft, IBM) had been thinking about scalable font technology support at the level of their respective operating systems since they realized that it would guarantee much better screen display, compared to pre-generated bitmaps which only look good at their design sizes, and unacceptably jagged at all others. For instance in the late 1980s Apple had developed an in-house scalable font technology, *Royal*, later renamed to TrueType.² The TrueType specification was public and already in 1991 native TrueType support appeared in Apple's Mac System 7 and Microsoft's Windows 3.1.

TrueType fonts use a different outline model from PostScript, and also the approach to hinting is different. The font instances contain both screen and printer font data in a single component. This makes the fonts easy to install. Although TrueType fonts support Unicode and can theoretically contain over 65.000 characters, they rarely feature more than some 220 characters. Moreover, TrueType font formats are platform-dependent.

1.1.3 Two competing technologies

Adobe reacted to the advent of TrueType by publishing in 1990 the PostScript Type 1 font format specification [1]. A few years later, it introduced the *Adobe Type Manager* (ATM) software, which scales PostScript Type 1 fonts for screen display, and supports imaging on non-PostScript printers.

Thus by the end of the 1990s there were two widely-used outline font specifications, TrueType, built into the operating systems used by most desktop computers, and PostScript Type 1, the de facto standard for the graphic arts and the publishing industry. Moreover, as time went by, the practical differences had begun to blur. On the one hand, support for TrueType became standard in PostScript 3, while on the other hand, besides native TrueType support, PostScript Type 1 rasterizing technology was incorporated into Windows 2000, Windows XP, and Mac OS X.

1.1.4 The best of two worlds: OpenType

The OpenType³ font format was jointly developed by Adobe and Microsoft to combine the best features of the TrueType and PostScript Type 1 technologies. It was first presented in 1996 and its use and support has been steadily increasing since about 2000.

OpenType fonts contain both the screen and printer font data in a single component. The OpenType format can contain either TrueType or PostScript font data. It supports expanded character sets (up to 65.000) and special typographic features. These may include various versions of figures (tabular, old-style, lining), small caps, ligatures, ordinals, and other extras. While OpenType allows type designers to build complex fonts, not many fonts take advantage of these possibilities. Most OpenType fonts available today are simply converted PostScript fonts, limited to 220 characters in a set.

OpenType fonts are platform independent and can thus be used on all operating systems.

¹The technology never really took off and since 2000 Adobe has abandoned developing multiple master fonts since most applications cannot handle them and for a large majority of users it often makes more economic sense to buy a fontset as multiple separate fonts. Adobe now concentrates on releasing OpenType fonts to replace their multiple master equivalents (e.g., the Minion and Myriad typefaces).

²See e.g., <http://developer.apple.com/fonts/>, and <http://www.microsoft.com/typography>.

³See Adobe's Web pages <http://store.adobe.com/type/opentype/main.html>, and <http://blogs.adobe.com/typblography/TT%20PS%20OpenType.pdf>, or Microsoft's Web page <http://www.microsoft.com/typography/OTSPEC/default.htm>.

1.2 PostScript Type 1 and TrueType: two different approaches

TrueType and PostScript Type 1 fonts use different mathematical representations to describe the curves defining the font outlines.¹ OpenType, being a superset, can have either kind of outlines.

TrueType describes its curves by quadratic B-splines, while PostScript Type 1 uses cubic Bézier curves. This means, in practice, that the shapes of real-world fonts tend to take more points in TrueType, even though the kind of mathematics used to describe the curves is simpler. Any quadratic spline can be converted to a cubic spline with essentially no loss. A cubic spline can be converted to a quadratic with arbitrary precision, but there will be a slight loss of accuracy in most cases. Thus it is easy to convert TrueType outlines to PostScript Type 1 outlines (the “Type 42” PostScript font format is a PostScript wrapper around a TrueType font for use in PostScript interpreters), harder to do the reverse.

The approach to hinting is different in both technologies. PostScript Type 1² takes a *declarative* approach and lets a *smart* PostScript interpreter do the work. It tells the rasterizer what features ought to be controlled, and the rasterizer interprets these using its own “intelligence” to decide how to do it. Therefore, when the PostScript interpreter is upgraded, the rasterization can be improved.

On paper, the hinting potential of TrueType³ should be superior to that of PostScript Type 1 fonts, since TrueType hints can do all that PostScript Type 1 can, and more. Indeed TrueType takes an *algorithmic* or programming approach and uses the very flexible and complete instructions set of the TrueType language. Thus TrueType puts all the hinting information into the font to control exactly how it will appear when rasterized. TrueType interpreters can be quite “dumb” and limit themselves to simply execute what they have been “instructed” to do. Thus, although a TrueType font developer can finetune what happens when a font is rasterized under different conditions, it requires serious effort, expertise, and high-end tools to actually take advantage of this greater hinting potential. As a result, high-quality TrueType fonts, which exploit the true potentials of TrueType hinting are quite rare. Moreover, when using complex hinting the introduction of a new rasterizer might require major changes to the TrueType code in order to be able to optimally display existing fonts.

PostScript Type 1 needs two separate files for its font data: one for the character outlines (`.pfb`), and the other for the metrics data (`.afm` on Linux, `.pfm` on Windows), containing character widths, kerning pairs, and a description of how to construct composites. TrueType fonts have all the data in a single file. Nevertheless this single TrueType font file is often twice larger than the two PostScript Type 1 files combined due to the presence in the TrueType fonts of extensive “hinting” instructions.

Generally speaking, PostScript Type 1 fonts have some advantages simply from being the longer-established standard, especially for serious graphic arts work. Service bureaus are standardized on, and have large investments in, PostScript Type 1 fonts. Most of the fonts which have “expert sets” of old style figures, extra ligatures, true small capitals and the like are in that format.

1.2.1 Interoperability

In principle one can mix TrueType and PostScript Type 1 fonts with the caveat that the TrueType and PostScript Type 1 instances of the fonts may not have exactly the same names on the given operating system. Indeed, the fact that fonts exist with identical menu names or PostScript Type 1 font names confuses the operating system or the application programs, with often unpredictable results.

Also, if using Windows, one may find that metrically-similar PostScript Type 1 fonts get substituted for the Windows TrueType system fonts at output time: *Times New Roman* becomes *Times Roman*, and *Arial* becomes *Helvetica*. Although the basic spacing of the substituted fonts is identical, their kerning pairs are not. This can cause text to reflow (i.e., line endings in a paragraph may differ) if one switches between two “almost identical” fonts if your typesetting program (e.g., T_EX) supports kerning pairs.

¹See <http://www.trueType.demon.co.uk/articles/ttvst1.htm>.

²See David Lemon’s *Basic Type 1 hinting* (<http://www.pyrus.com/downloads/hinting.pdf>).

³See the URL <http://www.microsoft.com/typography/hinting/tutorial.htm>, Vincent Connare’s *Basic hinting philosophies and TrueType instructions*.

Thus care must be taken to ensure that you use the correct font all through the complete production chain.

1.3 Unicode: the universal character encoding

Unicode is an international standard¹ for representing characters using a multi-byte platform-independent encoding for covering all the world languages (including some “artificial” ones, such as mathematical symbols and the international phonetic alphabet). Unicode deals with characters rather than glyphs. That is, it only deals with semantic rather than typographic distinctions (with a few exceptions for compatibility with existing standards). Therefore there is no place for glyph variants, such as unusual ligatures, old style numbers, or small caps within Unicode itself; the Unicode standard assumes that such distinctions will be made elsewhere. Therefore, font formats, which supports such distinctions, such as OpenType (see Section 1.4), need to be layered on top of Unicode. Alan Wood’s maintains a useful website (<http://www.alanwood.net/unicode/>) which describes numerous resources for Unicode and multilingual support in HTML, fonts, web browsers and other applications.

Most current operating systems (Linux, Mac OS X and Windows XP) have direct support for Unicode at the basic system level. For instance, apart from switching between different language keyboards, these operating systems offer means of directly accessing any Unicode character in any font (e.g., on Mac OS X via the *Character Palette* and on Microsoft Windows XP or Vista via the *Character Map* utility in *System Tools* in the *Accessories* submenu.)

1.4 OpenType

The OpenType font format was developed jointly by Microsoft and Adobe as an extension of the TrueType font format. OpenType addresses the following goals:

- supports PostScript Type 1 outlines and hints;
- supports TrueType tables and hints;
- supports advanced typographic features by way of new tables for glyph positioning and substitution;
- supports multiple platforms;
- supports international character sets by using Unicode;
- offers better protection for font data;
- features smaller file sizes to make font distribution more efficient.

Sometimes OpenType fonts are referred to as TrueType Open v.2.0 fonts. PostScript Type 1 data included in OpenType fonts may be directly rasterized or converted to the TrueType outline format for rendering, depending on which rasterizers have been installed in the host operating system. Users do not need to know which outlines are actually present. One can say that OpenType enters TrueType and PostScript Type 1 in a common wrapper. OpenType tables include the current TrueType tables plus some additional tables for advanced typographic features. The representation of PostScript Type 1 font software in an OpenType font uses Adobe’s Compact Font Format (CFF) with Type 2 charstrings, which is a more compact representation of the same information in PostScript Type 1 (a gain of about a factor of two, on average, when no glyphs and features are added).

¹The current version is 5.0 [7] and it has been defined by the members of the Unicode Consortium, which includes major computer corporations, software producers, database vendors, research institutions, international agencies, various user groups, and interested individuals, see <http://www.unicode.org>.

The OpenType format supports *features* equivalent to most of the advanced features of existing TrueType and PostScript formats, such as Adobe's CID technology for Asian fonts, and extended multilingual character sets. However, multiple master fonts are not part of the OpenType specification. OpenType fonts may contain more than 65,000 glyphs, which allows a single font file to contain many nonstandard glyphs, such as old-style figures, true small capitals, fractions, swashes, superiors, inferiors, titling letters, contextual and stylistic alternates, and a full range of ligatures. OpenType fonts thus offers rich linguistic support combined with advanced typographic control. Feature-rich Adobe OpenType fonts are often distinguished by the word "Pro," being part of the font name. OpenType fonts can be installed and used alongside PostScript Type 1 and TrueType fonts.

OpenType, which is based on Unicode, significantly simplifies font management and the publishing process by ensuring that all of the required glyphs for a document are contained in one cross-platform font file throughout the workflow.

The text model of OpenType is that applications store text using the underlying Unicode characters, and apply formatting to get at the specific desired glyphs. In addition to the Unicode mapping of default glyphs, the font has OpenType layout tables which tell it which glyphs to use when other forms are desired instead, such as small caps or swashes. These tables also specify which glyphs should turn into ligatures, or when a script font needs different glyphs for a letter when it is at the beginning, middle or end of a word, or is a word by itself.

Having the transformations distinct from the underlying text enables table-driven automatic glyph substitution, which does not need to be one for one; one glyph can be substituted for several (such as the "ffi" ligature, which remembers that the underlying text contains the characters "f-f-i" in searching), or multiple glyphs can be substituted for a single one. Glyph substitution can be context sensitive, or it can be activated by explicit user demand. This feature might not appear essential for Latin-based languages, such as Spanish and English, but it becomes mandatory for proper typesetting of languages that use "complex scripts", such as Arabic or the Indic languages, since having letters take different forms based on their position in the word is a basic part of how Arabic works.

OpenType layout features can be used to position or substitute glyphs. For any character, there is a default glyph and positioning behavior. The application of layout features to one or more characters may change the positioning, or substitute a different glyph.

There are several advantages of using a large OpenType font over currently available "expert sets" and "alternates". First, one only has to deal with one font file, rather than being cluttered with a whole set of supplemental fonts. Second, there can be kerning between glyphs that might otherwise have been in separate fonts. Finally, the user can turn on ligatures, smallcaps, or old-style figures, much like bold or italic styling, without switching fonts.

Historically, some of the highest quality typefaces have included different designs for different print sizes. Rather than using its multiple masters technology, most of Adobe's OpenType fonts now include four optical size variations: caption, regular, subhead and display. Called "Opticals," these variations have been optimised for use at specific point sizes. Although the exact intended sizes vary by family, the general size ranges include: caption (6–8 point), regular (9–13 point), subhead (14–24 point) and display (25–72 point).

1.4.1 OpenType tables

OpenType font files contain tables that contain either TrueType or PostScript outline font data and the data in these tables are used by rendering programs to render the TrueType or PostScript glyphs. Moreover, some of the data is independent of the particular outline format used.¹

OpenType fonts first contain a number of *required* tables.

¹The structure of an OpenType font file is described at the URL <http://www.microsoft.com/typography/otspec/otff.htm>; a short description of the contents of the tables is at the URL <http://www.microsoft.com/typography/otspec/recom.htm>.

cmap	Character to glyph mapping	maxp	Maximum profile
head	Font header	name	Naming table
hhea	Horizontal header	OS/2	OS/2 and Windows specific metrics
hmtx	Horizontal metrics	post	PostScript information

For OpenType fonts based on TrueType outlines, the following tables are used:

cvt	Control Value Table	glyf	Glyph data	prep	CVT Program
fpgm	Font program	loca	Index to location		

For OpenType fonts based on PostScript another set of tables containing data specific to PostScript fonts are used instead of the tables listed above:

cff	PostScript font program (compact font format)
vorg	Vertical Origin

OpenType fonts may contain bitmaps of glyphs, in addition to outlines. Hand-tuned bitmaps are especially useful in OpenType fonts for representing complex glyphs at very small sizes. If a bitmap for a particular size is provided in a font, it will be used by the system instead of the outline when rendering the glyph. For OpenType fonts containing bitmap glyphs three tables are available:

EBDT	Embedded bitmap data
EBLC	Embedded bitmap location data
EBSC	Embedded bitmap scaling data

Finally, advanced typography, vertical typesetting and other special functions are supported with the following tables:

BASE	Baseline data	hdmx	Horizontal device metrics
GDEF	Glyph definition data	kern	Kerning
GPOS	Glyph positioning data	LTSH	Linear threshold data
GSUB	Glyph substitution data	PCLT	PCL 5 data
JSTF	Justification data	VDMX	Vertical device metrics
DSIG	Digital signature	vhea	Vertical Metrics header
gasp	Grid-fitting/Scan-conversion	vmtx	Vertical Metrics

Furthermore, OpenType fonts use a set of script, language and feature tags to structure the information in their tables.

Script tags identify the scripts represented in an OpenType font. Each script corresponds to a contiguous character code range in Unicode. Script tags are four-byte character strings composed of up to four letters in the ASCII characters range 0x20–0x7E, padding with blanks (0x20) if required. A list of scripts and their tags follows.¹

df1t	Default	cans	Canadian Syllabics	gujr	Gujarati
arab	Arabic	cher	Cherokee	guru	Gurmukhi
armn	Armenian	cyrl	Cyrillic	jamo	Hangul Jamo
beng	Bengali	deva	Devanagari	hang	Hangul
bopo	Bopomofo	ethi	Ethiopic	hani	CJK Ideographic
brai	Braille	geor	Georgian	hebr	Hebrew
byzm	Byzantine Music	grek	Greek	kana	Hiragana

¹See <http://www.microsoft.com/typography/otspec/scripttags.htm> for an up-to-date list.

knda	Kannada	mymr	Myanmar	telu	Telugu
kana	Katakana	ogam	Ogham	thaa	Thaana
khmr	Khmer	orya	Oriya	thai	Thai
lao	Lao	runr	Runic	tibt	Tibetan
latn	Latin	sinh	Sinhala	yi	Yi
mlym	Malayalam	syrc	Syriac		
mong	Mongolian	taml	Tamil		

When the table with the list of scripts is searched for a script, and no entry is found, and there exists an entry for the `DFLT` script, then this entry must be used. Furthermore, the default script can only contain a single, default, language.

Language system tags identify the language systems supported in an OpenType font. What is meant by a “language system” in this context is a set of typographic conventions for how text in a given script should be presented. Such conventions may be associated with particular languages, with particular genres of usage, with different publications, and other such factors. For example, particular glyph variants for certain characters may be required for particular languages, or for phonetic transcription or mathematical notation.

Note that two or more languages may follow the same conventions or that more than one set of typographic conventions can apply to a given language. Therefore language system tags do not correspond in a one-to-one manner with languages.¹

Language system tags are four-byte character strings composed of up to four characters in the ASCII characters range `0x20–0x7E`, padding with blanks (`0x20`) if required. A list of languages and their language system tags follows.

df1t	Default	BAL	Balkar	BRM	Burmese
ABA	Abaza	BAU	Baule	BSH	Bashkir
ABK	Abkhazian	BBR	Berber	BTI	Beti
ADY	Adyghe	BCH	Bench	CAT	Catalan
AFK	Afrikaans	BCR	Bible Cree	CEB	Cebuano
AFR	Afar	BEL	Belarussian	CHE	Chechen
AGW	Agaw	BEM	Bemba	CHG	Chaha Gurage
ALT	Altai	BEN	Bengali	CHH	Chattisgarhi
AMH	Amharic	BGR	Bulgarian	CHI	Chichewa
APPH	Phonetic transcription (Americanist conventions)	BHI	Bhili	CHK	Chukchi
ARA	Arabic	BHO	Bhojpuri	CHP	Chipewyan
ARI	Aari	BIK	Bikol	CHR	Cherokee
ARK	Arakanese	BIL	Bilen	CHU	Chuvash
ASM	Assamese	BKF	Blackfoot	CMR	Comorian
ATH	Athapaskan	BLI	Balochi	COP	Coptic
AVR	Avar	BLN	Balante	CRE	Cree
AWA	Awadhi	BLT	Balti	CRR	Carrier
AYM	Aymara	BMB	Bambara	CRT	Crimean Tatar
AZE	Azeri	BML	Bamileke	CSL	Church Slavonic
BAD	Badaga	BRE	Breton	CSY	Czech
BAG	Baghelkhandi	BRH	Brahui	DAN	Danish
		BRI	Braj Bhasha	DAR	Dargwa

¹See <http://www.microsoft.com/typography/otspec/scripttags.htm> for an up-to-date list of language tags and the correspondence to the ISO 639 codes, which identify individual languages as well as for certain collections of languages.

DCR	Woods Cree	GRO	Garó	KHK	Khanty-Kazim
DEU	German (Standard)	GUA	Guarani	KHM	Khmer
DGR	Dogri	GUJ	Gujarati	KHS	Khanty-Shurishkar
DHV	Dhivehi	HAI	Haitian	KHV	Khanty-Vakhi
DJR	Djerma	HAL	Halam	KHW	Khowar
DNG	Dangme	HAR	Harauti	KIK	Kikuyu
DNK	Dinka	HAU	Hausa	KIR	Kirghiz
DUN	Dungan	HAW	Hawaiian	KIS	Kisii
DZN	Dzongkha	HBN	Hammer-Banna	KKN	Kokni
EBI	Ebira	HIL	Hiligaynon	KLM	Kalmyk
ECR	Eastern Cree	HIN	Hindi	KMB	Kamba
EDO	Edo	HMA	High Mari	KMN	Kumaoni
EFI	Efik	HND	Hindko	KMO	Komo
ELL	Greek	HO	Ho	KMS	Komso
ENG	English	HRI	Harari	KNR	Kanuri
ERZ	Erzya	HRV	Croatian	KOD	Kodagu
ESP	Spanish	HUN	Hungarian	KOK	Konkani
ETI	Estonian	HYE	Armenian	KON	Kikongo
EUQ	Basque	IBO	Igbo	KOP	Komi-Permyak
EVK	Evenki	IJO	Ijo	KOR	Korean
EVN	Even	ILO	Ilokano	KOZ	Komi-Zyrian
EWE	Ewe	IND	Indonesian	KPL	Kpelle
FAN	French Antillean	ING	Ingush	KRI	Krio
FAR	Farsi	INU	Inuktitut	KRK	Karakalpak
FIN	Finnish	IPPH	Phonetic transcription (IPA conventions)	KRL	Karelian
FJI	Fijian	IRI	Irish	KRM	Karaim
FLE	Flemish	IRT	Irish Traditional	KRN	Karen
FNE	Forest Nenets	ISL	Icelandic	KRT	Koorete
FON	Fon	ISM	Inari Sami	KSH	Kashmiri
FOS	Faroese	ITA	Italian	KSI	Khasi
FRA	French (Standard)	IWR	Hebrew	KSM	Kildin Sami
FRI	Frisian	JAN	Japanese	KUI	Kui
FRL	Friulian	JAV	Javanese	KUL	Kulvi
FTA	Futa	JII	Yiddish	KUM	Kumyk
FUL	Fulani	JUD	Judezmo	KUR	Kurdish
GAD	Ga	JUL	Jula	KUU	Kurukh
GAE	Gaelic	KAB	Kabardian	KUY	Kuy
GAG	Gagauz	KAC	Kachchi	KYK	Koryak
GAL	Galician	KAL	Kalenjin	LAD	Ladin
GAR	Garshuni	KAN	Kannada	LAH	Lahuli
GAW	Garhwali	KAR	Karachay	LAK	Lak
GEZ	Geez	KAT	Georgian	LAM	Lambani
GIL	Gilyak	KAZ	Kazakh	LAO	Lao
GMZ	Gumuz	KEB	Kebena	LAT	Latin
GON	Gondi	KGE	Khutsuri Georgian	LAZ	Laz
GRN	Greenlandic	KHA	Khakass	LCR	L-Cree
				LDK	Ladakhi
				LEZ	Lezgi
				LIN	Lingala
				LMA	Low Mari

LMB	Limbu	NDG	Ndonga	SEL	Selkup
LMW	Lomwe	NEP	Nepali	SGO	Sango
LSB	Lower Sorbian	NEW	Newari	SHN	Shan
LSM	Lule Sami	NHC	Norway House Cree	SIB	Sibe
LTH	Lithuanian	NIS	Nisi	SID	Sidamo
LUB	Luba	NIU	Niuean	SIG	Silte Gurage
LUG	Luganda	NKL	Nkole	SKS	Skolt Sami
LUH	Luhya	NLD	Dutch	SKY	Slovak
LUO	Luo	NOG	Nogai	SLA	Slavey
LVI	Latvian	NOR	Norwegian	SLV	Slovenian
MAJ	Majang	NSM	Northern Sami	SML	Somali
MAK	Makua	NTA	Northern Tai	SMO	Samoan
MAL	Malayalam Traditional	NTO	Esperanto	SNA	Sena
MAN	Mansi	NYN	Nynorsk	SND	Sindhi
MAR	Marathi	OCR	Oji-Cree	SNH	Sinhalese
MAW	Marwari	OJB	Ojibway	SNK	Soninke
MBN	Mbundu	ORI	Oriya	SOG	Sodo Gurage
MCH	Manchu	ORO	Oromo	SOT	Sotho
MCR	Moose Cree	OSS	Ossetian	SQI	Albanian
MDE	Mende	PAA	Palestinian Aramaic	SRB	Serbian
MEN	Me'en	PAL	Pali	SRK	Saraiki
MIZ	Mizo	PAN	Punjabi	SRR	Serer
MKD	Macedonian	PAP	Palpa	SSL	South Slavey
MLE	Male	PAS	Pashto	SSM	Southern Sami
MLG	Malagasy	PGR	Polytonic Greek	SUR	Suri
MLN	Malinke	PIL	Pilipino	SVA	Svan
MLR	Malayalam Reformed	PLG	Palaung	SVE	Swedish
MLY	Malay	PLK	Polish	SWA	Swadaya Aramaic
MND	Mandinka	PRO	Provençal	SWK	Swahili
MNG	Mongolian	PTG	Portuguese	SWZ	Swazi
MNI	Manipuri	QIN	Chin	SXT	Sutu
MNK	Maninka	RAJ	Rajasthani	SYR	Syriac
MNX	Manx Gaelic	RBU	Russian Buriat	TAB	Tabasaran
MOK	Moksha	RCR	R-Cree	TAJ	Tajiki
MOL	Moldavian	RIA	Riang	TAM	Tamil
MON	Mon	RMS	Rhaeto-Romanic	TAT	Tatar
MOR	Moroccan	ROM	Romanian	TCR	TH-Cree
MRI	Maori	ROY	Romany	TEL	Telugu
MTH	Maithili	RSY	Rusyn	TGN	Tongan
MTS	Maltese	RUA	Ruanda	TGR	Tigre
MUN	Mundari	RUS	Russian	TGY	Tigrinya
NAG	Naga-Assamese	SAD	Sadri	THA	Thai
NAN	Nanai	SAN	Sanskrit	THT	Tahitian
NAS	Naskapi	SAT	Santali	TIB	Tibetan
NCR	N-Cree	SAY	Sayisi	TKM	Turkmen
NDB	Ndebele	SEK	Sekota	TMN	Temne

TNA	Tswana	URD	Urdu	YAK	Yakut
TNE	Tundra Nenets	USB	Upper Sorbian	YBA	Yoruba
TNG	Tonga	UYG	Uyghur	YCR	Y-Cree
TOD	Todo	UZB	Uzbek	YIC	Yi Classic
TRK	Turkish	VEN	Venda	YIM	Yi Modern
TSG	Tsonga	VIT	Vietnamese	ZHP	Chinese Phonetic
TUA	Turoyo Aramaic	WAG	Wagdi	ZHS	Chinese Simplified
TUL	Tulu	WA	Wa	ZHT	Chinese Traditional
TUV	Tuvin	WCR	West-Cree	ZND	Zande
TWI	Twi	WEL	Welsh	ZUL	Zulu
UDM	Udmurt	WLF	Wolof		
UKR	Ukrainian	XHS	Xhosa		

1.4.2 OpenType features

Features provide information about how to use the glyphs in an OpenType or TrueType font to render a script or language. For example, an Arabic font might have a feature for substituting initial glyph forms, and a Kanji font might have a feature for positioning glyphs vertically. All OpenType Layout features define data for glyph substitution, glyph positioning, or both.

Each OpenType Layout feature has a feature tag that identifies its typographic function and effects. By examining a feature's tag, a text-processing client can determine what a feature does and decide whether to implement it. All tags are four-byte character strings composed of a limited set of ASCII characters (range 0x20–0x7E).

A feature definition does not necessarily provide all the information required to properly implement glyph substitution or positioning actions. Often, a text-processing client may need to supply additional data¹ In all cases, the text-processing client is responsible for applying, combining, and arbitrating among features and rendering the result.

The list of features registered by Microsoft together with a short description follows.²

aalt	Access All Alternates	clig	Contextual Ligatures	fin2	Terminal Forms #2
abvf	Above-base Forms	cpSP	Capital Spacing	fin3	Terminal Forms #3
abvm	Above-base Mark Positioning	cswh	Contextual Swash	finA	Terminal Forms
abvs	Above-base Substitutions	 curs	Cursive Positioning	frac	Fractions
afrC	Alternative Fractions	c2sc	Small Capitals From Capitals	fwid	Full Widths
akhn	Akhands	c2pc	Petite Capitals From Capitals	half	Half Forms
blwf	Below-base Forms	dist	Distances	haln	Halant Forms
blwm	Below-base Mark Positioning	dlig	Discretionary Ligatures	halt	Alternate Half Widths
blws	Below-base Substitutions	dnom	Denominators	hist	Historical Forms
calt	Contextual Alternates	expt	Expert Forms	hkna	Horizontal Kana Alternates
case	Case-Sensitive Forms	falt	Final Glyph on Line Alternates	hlig	Historical Ligatures
ccmp	Glyph Composition and Decomposition			hngl	Hangul
				hojo	Hojo Kanji Forms (JIS X 0212-1990 Kanji Forms)

¹As an example let us consider the *init* feature whose function is to provide initial glyph forms. Nothing in the feature's lookup tables indicates when or where to apply this feature during text processing. Hence, to correctly use this feature in Arabic text where initial glyph forms appear at the beginning of words, text-processing clients must be able to identify the first glyph position in each word before making the glyph substitution.

²More details about each feature are available at the Microsoft OpenType site <http://www.microsoft.com/typography/otspec/featuretags.htm>, or Adobe developers' site http://partners.adobe.com/public/developer/opentype/index_tag3.html

hwid	Half Widths	palt	Proportional Alternate Widths	ss12	Stylistic Set 12
init	Initial Forms	pcap	Petite Capitals	ss13	Stylistic Set 13
iso1	Isolated Forms	pnum	Proportional Figures	ss14	Stylistic Set 14
ital	Italics	pref	Pre-Base Forms	ss15	Stylistic Set 15
jalt	Justification Alternates	pres	Pre-base Substitutions	ss16	Stylistic Set 16
jp78	JIS78 Forms	pstf	Post-base Forms	ss17	Stylistic Set 17
jp83	JIS83 Forms	psts	Post-base Substitutions	ss18	Stylistic Set 18
jp90	JIS90 Forms	pwid	Proportional Widths	ss19	Stylistic Set 19
jp04	JIS2004 Forms	qwid	Quarter Widths	ss20	Stylistic Set 20
kern	Kerning	rand	Randomize	subs	Subscript
lfbd	Left Bounds	rlig	Required Ligatures	sup	Superscript
liga	Standard Ligatures	rphf	Reph Forms	swsh	Swash
ljmo	Leading Jamo Forms	rtbd	Right Bounds	titl	Titling
lnum	Lining Figures	rtla	Right-to-left alternates	tjmo	Trailing Jamo Forms
loc1	Localized Forms	ruby	Ruby Notation Forms	tnam	Traditional Name Forms
mark	Mark Positioning	salt	Stylistic Alternates	tnum	Tabular Figures
med2	Medial Forms #2	sinf	Scientific Inferiors	trad	Traditional Forms
medi	Medial Forms	size	Optical size	twid	Third Widths
mgrk	Mathematical Greek	smcp	Small Capitals	unic	Uppercase
mkmk	Mark to Mark Positioning	smp1	Simplified Forms	valt	Alternate Vertical Metrics
mset	Mark Positioning via Substitution	ss01	Stylistic Set 1	vatu	Vattu Variants
na1t	Alternate Annotation Forms	ss02	Stylistic Set 2	vert	Vertical Writing
nlck	NLC Kanji Forms	ss03	Stylistic Set 3	vhal	Alternate Vertical Half Metrics
nukt	Nukta Forms	ss04	Stylistic Set 4	vjmo	Vowel Jamo Forms
numr	Numerators	ss05	Stylistic Set 5	vkna	Vertical Kana Alternates
onum	Oldstyle Figures	ss06	Stylistic Set 6	vkern	Vertical Kerning
opbd	Optical Bounds	ss07	Stylistic Set 7	vpal	Proportional Alternate Vertical Metrics
ordn	Ordinals	ss08	Stylistic Set 8	vrt2	Vertical Alternates and Rotation
ornm	Ornaments	ss09	Stylistic Set 9	zero	Slashed Zero
		ss10	Stylistic Set 10		
		ss11	Stylistic Set 11		

1.4.3 OpenType support today

As an example of how publishing applications can exploit OpenType's layout features we can look at OpenType support in Adobe's Illustrator, InDesign and Photoshop¹ programs. These include automatic substitution by alternate glyphs in an OpenType Pro font (ligatures, small capitals, and proportional old-style figures, vertical shift of punctuation in an all-caps setting). Moreover, any alternate glyphs in OpenType fonts may be selected manually via the *Insert Character* palette (see Figure 1.1 on the facing page). These OpenType Pro fonts offer a full range of accented characters to support all central and eastern European languages, and many of them also contain support for the Cyrillic and Greek alphabets.

Feature support across Microsoft's Office applications exists for those features that are necessary for language support, such as contextual substitutions for Arabic—and only in the languages which require them (e.g., Word 2003 does contextual substitutions for Arabic, but not for English).

¹See <http://www.adobe.com/products/XXX/main.htm>, where XXX stands for *illustrator*, *indesign*, and *photoshop*, respectively.

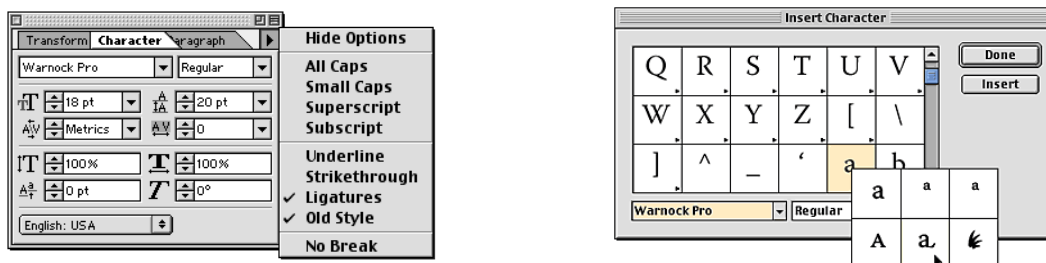


Figure 1.1: Using OpenType’s advanced typographic features in Adobe InDesign. Left: selection of automatic substitution of ligatures and old-style figures on a menu. Right: select and insert any alternate glyph *Insert Character* palette.

Openoffice on all supported platforms has a somewhat similar approach to Microsoft’s Office suite in that it allows one to use the characters present in the font but does not really present an interface to the advanced typographic features (see Figure 1.2 on the next page).

That leaves us with the availability of the fonts themselves. Around the year 2000 there were only a handful of OpenType fonts, and almost all of them were from Adobe. Nowadays, there are thousands available from over two dozen font foundries. For instance, the entire Adobe Type Library of over 2,200 fonts has been translated into the OpenType format, URW has released over 1,000 OpenType fonts, and other large foundries, such as Linotype and Agfa Monotype, as well as most smaller foundries, are also creating OpenType fonts. Most of Microsoft’s system fonts, and Apple’s Japanese system fonts, are OpenType. Similarly, OpenType is being embraced by major type foundries for non-alphabetic scripts, such as Chinese and Japanese.

However, it is not enough for a font to be in the OpenType format to be sure that it has extended language support or extra typographic features. Therefore, before purchasing, you should examine the features present in a font.¹ To inspect a font that you already have on your Microsoft Windows system, you can install the *Font Properties Extension* from Microsoft. This add-on allows you to right-click on a font to display a much expanded set of properties, which includes language support and OpenType layout features (see Figure 1.3 on page 15).

1.4.4 Interrogating OpenType fonts

Eddie Kohler’s `otfinfo` program² prints information about an OpenType font.

```
> otfinfo --help
'Otfinfo' reports information about an OpenType font to standard output.
Options specify what information to print.

Usage: otfinfo [-sfzpg] [OTFFILES...]

Query options:
  -s, --scripts           Report font's supported scripts.
  -f, --features          Report font's GSUB/GPOS features.
  -z, --optical-size      Report font's optical size information.
  -p, --postscript-name   Report font's PostScript name.
  -a, --family           Report font's family name.
```

¹In the case of Adobe, where currently not all fonts released in OpenType format have significant added features or extended language support, you browse all fonts in the *Adobe Type Library* from the URL <http://store.adobe.com/type/main.html>, so that you can inspect the font you are interested in. Other font vendors offer similar possibilities.

²Part of his `lcdf` tools, see www.lcdf.org/type/.

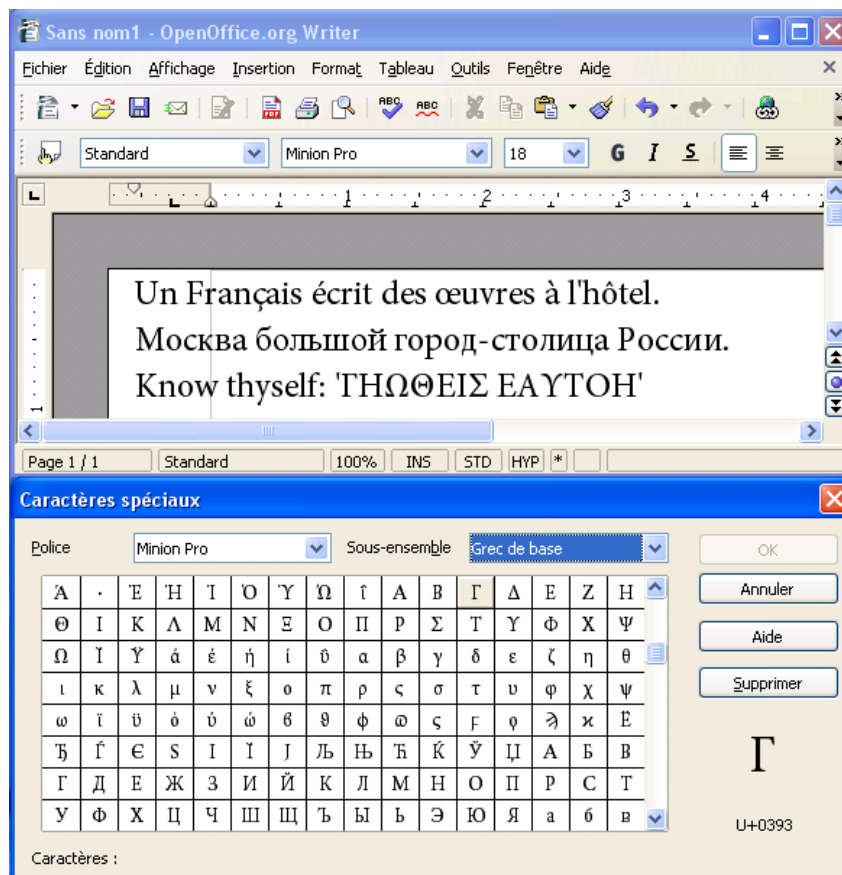


Figure 1.2: OpenType Unicode support in OpenOffice. The top panel shows text in various alphabets and the bottom panel the characters available in the Greek part of font layout.

```
-v, --font-version      Report font's version information.
-i, --info              Report font's names and designer/vendor info.
-g, --glyphs           Report font's glyph names.
-t, --tables           Report font's OpenType tables.
```

Other options:

```
--script=SCRIPT[.LANG] Set script used for --features [latn].
-V, --verbose          Print progress information to standard error.
-h, --help            Print this message and exit.
-q, --quiet           Do not generate any error messages.
--version            Print version number and exit.
```

```
> otfinfo --info texmf-commercial/fonts/opentype/adobe/minionpro-regular.otf
Family:                Minion Pro
Subfamily:             Regular
Full name:             Minion Pro
PostScript name:      MinionPro-Regular
Version:              Version 2.012;PS 002.000;Core 1.0.38;makeotf.lib1.6.6565
Unique ID:            2.012;ADBE;MinionPro-Regular
```

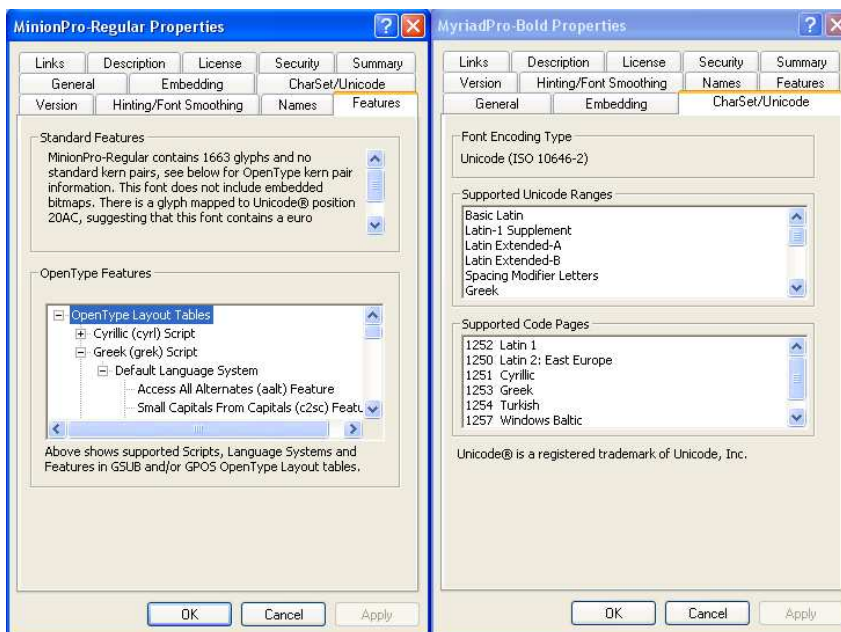


Figure 1.3: Microsoft’s *Fonts Extension* utility displays OpenType features for MinionPro-Regular and the supported Character sets for MyriadPro-Bold when you right-click on the font (This utility, `ttfont`, adds several new property tabs to the standards properties dialog box, such as information relating to font origination and copyright, the type sizes to which hinting and smoothing are applied, and the code pages supported by extended character. It can be downloaded from <http://www.microsoft.com/Typography/TrueTypeProperty21.msp>.)

```
Designer:           Robert Slimbach
Vendor URL:         http://www.adobe.com/type/
Trademark:         Minion is either a ...
Copyright:         © 2000, 2002, 2004 ...
License URL:       http://www.adobe.com/type/legal.html
```

```
> otfont --script texmf-commercial/fonts/opentype/adobe/minionpro-regular.otf
cyril      Cyrillic      latn.DEU      Latin/German (Standard)
grek       Greek          latn.MOL      Latin/Moldavian
latn       Latin          latn.ROM      Latin/Romanian
latn.AZE   Latin/Azeri    latn.SRB      Latin/Serbian
latn.CRT   Latin/Crimean Tatar latn.TRK      Latin/Turkish
```

```
> otfont --tables texmf-commercial/fonts/opentype/adobe/minionpro-regular.otf
  64 BASE          54 head
132417 CFF         36 hhea
 5228 DSIG        6652 hmtx
40074 GPOS         6 maxp
13872 GSUB        1533 name
  96 OS/2         32 post
4048 cmap
```

```
> otfont --features texmf-commercial/fonts/opentype/adobe/minionpro-regular.otf
```

aalt	Access All Alternates	c2sc	Small Capitals From Capitals
case	Case-Sensitive Forms	csp	Capital Spacing
dlig	Discretionary Ligatures	dnom	Denominators
fin	Terminal Forms	frac	Fractions
hist	Historical Forms	kern	Kerning
liga	Standard Ligatures	lnum	Lining Figures
numr	Numerators	onum	Oldstyle Figures
ordn	Ordinals	ornm	Ornaments
pnum	Proportional Figures	salt	Stylistic Alternates
sinf	Scientific Inferiors	size	Optical Size
smcp	Small Capitals	ss01	Stylistic Set 1
ss02	Stylistic Set 2	sup	Superscript
tnum	Tabular Figures	zero	Slashed Zero

Just van Rossum's `ttx` utility¹ can decompile the contents of an OpenType font and output it in XML format. This comes in handy if you want to study the contents of a given font (e.g., its tables) or (slightly) modify it.

```
> ttx --help
usage: ttx [options] inputfile1 [... inputfileN]

TTX 2.0b1 -- From OpenType To XML And Back

If an input file is a TrueType or OpenType font file, it will be
dumped to an TTX file (an XML-based text format).
If an input file is a TTX file, it will be compiled to a TrueType
or OpenType font file.

Output files are created so they are unique: an existing file is
never overwritten.

General options:
-h Help: print this message
-d <outputfolder> Specify a directory where the output files are
to be created.
-v Verbose: more messages will be written to stdout about what
is being done.

Dump options:
-l List table info: instead of dumping to a TTX file, list some
minimal info about each table.
-t <table> Specify a table to dump. Multiple -t options
are allowed. When no -t option is specified, all tables
will be dumped.
-x <table> Specify a table to exclude from the dump. Multiple
-x options are allowed. -t and -x are mutually exclusive.
-s Split tables: save the TTX data into separate TTX files per
table and write one small TTX file that contains references
to the individual table dumps. This file can be used as
input to ttx, as long as the table files are in the
same directory.
-i Do NOT disassemble TT instructions: when this option is given,
all TrueType programs (glyph programs, the font program and the
```

¹Written in Python and part of the FontTools toolset (sourceforge.net/projects/fonttools).

pre-program) will be written to the TTX file as hex data instead of assembly. This saves some time and makes the TTX file smaller.

Compile options:

- m Merge with TrueType-input-file: specify a TrueType or OpenType font file to be merged with the TTX file. This option is only valid when at most one TTX file is specified.
- b Don't recalc glyph bounding boxes: use the values in the TTX file as-is.

Thus, to decompile a font `myfont.otf` just specify:

```
> ttx myfont.otf
```

This will write a file `myfont.ttx` in the directory where the font file resides. If you are only interested in two tables (e.g., GSUB and GPOS), specify them on the command line:

```
> ttx -t GSUB -t GPOS myfont.otf
```

To convert an XML file `myfont.ttx` back into an OpenType or TrueType file is similarly easy:

```
> ttx myfont.ttx
```

If you want to introduce modifications (e.g., given in XML format in the file `myfontmods.ttx`) into an OpenType file, use the `-m` option, as follows:

```
> ttx -m myfont.otf myfontmods.ttx
```

A more explicit example with the font MinionPro follows.

```
> ttx -l /texlive/2007/texmf-commercial/fonts/opentype/adobe/minionpro-regular.otf  
Listing table info for
```

```
"/texlive/2007/texmf-commercial/fonts/opentype/adobe/minionpro-regular.otf":  
tag      checksum      length  offset  tag      checksum      length  offset  
----      -  
BASE     0x086729a7      64     199052  CFF     0x101232c2     132417  6032  
DSIG     0x446dbd94     5228    199116  GPOS    0xx71552700     40074   158976  
GSUB     0xx3bf7bcba    13872   145104  OS/2    0x40e57e9f      96      320  
cmap     0x0cedc8f1     4048    1952   head    0xx2167aded      54      220  
hhea     0x09140bb5      36      276   hmtx    0xx37425493     6652   138452  
maxp     0x067f5000      6       312   name    0x3cf7b183     1533    416  
post     0x0x47ffce      32     6000
```

```
ttx -d. -t head /texlive/2007/texmf-commercial/fonts/opentype/adobe/minionpro-regular.otf  
Dumping "/texlive/2007/texmf-commercial/fonts/opentype/adobe/minionpro-regular.otf"  
to "./minionpro-regular.ttx"...
```

```
Dumping 'head' table...
```

```
> less ./minionpro-regular.ttx
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

```
<ttFont sfntVersion="OTTO" ttLibVersion="2.0b1">
```

```
<head>
```

```
<!-- Most of this table will be recalculated by the compiler -->
```

```
<tableVersion value="1.0"/>
```

```
<fontRevision value="2.0119934082"/>
```

```
<checksumAdjustment value="-0x107d913c"/>
<magicNumber value="0x5f0f3cf5"/>
<flags value="00000000 00000011"/>
<unitsPerEm value="1000"/>
<created value="Tue Jun 29 11:41:10 2004"/>
<modified value="Tue Jun 29 11:41:10 2004"/>
<xMin value="-290"/>
<yMin value="-360"/>
<xMax value="1684"/>
<yMax value="989"/>
<macStyle value="00000000 00000000"/>
<lowestRecPPEM value="3"/>
<fontDirectionHint value="2"/>
<indexToLocFormat value="0"/>
<glyphDataFormat value="0"/>
</head>
</ttFont>
```

For reasons of efficiency TrueType and OpenType font instances can be grouped into “collection” (.ttc), so that different fonts can share common tables to describe glyphs. Some programs are not able to extract the various font components from such a collection. To help with this problem a small utility, `ttc2ttf`, exists to extract the font instances from a collection.

CHAPTER 2

X_YTEX: T_EX meets OpenType and Unicode

2.1 X _Y TEX: a historical introduction and some basics	21
2.2 X _Y TEX: typesetting with glyphs, characters and fonts	23
2.3 Supplementary commands introduced by X _Y TEX	33
2.4 fontspec	43
2.5 X _Y TEX and other engines	47

X_YTEX is a typesetting system based on a merger of e-T_EX with Unicode and modern font technologies. Jonathan Kew is the main developer behind X_YTEX. X_YTEX's main aim is to deal with the complexities (notice the colored parts on the characters in Figure 2.1) needed to typeset texts in the various scripts used in the world (Figure 2.2 on the next page), in particular in Asia (Figure 2.3 on the following page).



Figure 2.1: Complexities when dealing with various languages

We start the chapter with an introduction, a short history and an overview of the basic operating principles of XeTeX (Section 2.1). XeTeX’s character/glyph model, its typesetting algorithm and the way it handles fonts is the subject of Section 2.2.

Section 2.3 presents in detail the supplementary commands introduced by XeTeX, in particular its extension to TeX’s `\font` command to take full advantage of the possibilities of the OpenType fonts. A \LaTeX interface to XeTeX’s font handling is presented in Section 2.4.

2.1 XeTeX: a historical introduction and some basics

XeTeX¹ was developed at SIL² by its author Jonathan Kew. One of XeTeX’s important aims is to allow the TeX engine to directly use fonts available on the operating system. Technically this is implemented by augmenting TeX’s `\font` command so that it asks the host operating system to locate a given font (using its real name, as known to the operating system, not some cryptic filename, e.g., à la Berry) in whatever font collection available. This means that all fonts known on a system and available to the user interface become usable for typesetting in XeTeX and with the same names. Hence it is no longer necessary to run any TeX-specific procedures (e.g., `fontinst`, or apply one of the recipes described earlier in this chapter). When XeTeX is instructed to use a font, it locates the actual font file itself (it can handle all three variants OpenType, PostScript Type 1, and TrueType), and no longer needs a `.tfm` file. XeTeX’s paragraph building routine thus obtains metric information about the character glyphs directly from the font file. In addition, it has to take care of the complexities of mapping characters to glyphs, particularly in cursive and non-Latin scripts. Therefore, XeTeX does not build its paragraphs from lists of characters, but from “words”, each of which consists of a whole run of consecutive characters in a given font. Linguistical and typographical transformations and effects are delegated to the appropriate “layout engine” (XeTeX has interfaces to ATSUI,³ ICU,⁴ and SIL’s Graphite).⁵ The result is an array of glyphs and their positions that represent words as laid out using the current font. From this list of words, which are interleaved with glue, penalties, etc., a paragraph is built. Of course, when hyphenation is required, “words” may have to be taken apart and reassembled afterwards using possible break positions. Nevertheless the basic idea remains: collect runs of characters, hand them down as complete units to a font rendering library, which is capable of handling the layout at the level of the individual glyphs.

XeTeX works with an extended version of the existing `dvipdfmx` PDF driver, where the help of Jin-Hwan Cho has to be acknowledged. Akira Kakuto’s `W32tex` (<http://www.fsci.fuk.kindai.ac.jp/kakuto/win32-ptex>) has contributed a lot to make XeTeX available on Microsoft Windows. Ross Moore has worked on graphics and color drivers, while Miyata Shigeru has improved the handling of vertical text and CJK support in both XeTeX itself and the driver, and provides support for PSTricks graphics.

¹This section is based on an interview with XeTeX’s author Jonathan Kew. For the full text of the interview see <http://tug.org/interviews/interview-files/jonathan-kew.html>.

²SIL (initially known as the *Summer Institute of Linguistics*, see <http://www.sil.org> for more information) was created in 1934. It now has about 5,000 collaborators coming from over 60 countries. SIL’s main activity is the linguistic investigation of some 1,800 languages spoken by more than a billion people in more than 70 countries. In particular, SIL publishes *Ethnologue, languages of the world* (<http://www.ethnologue.com/>), a book which describes 6912 languages spoken on earth.

³Apple Type Services for Unicode Imaging is the technology behind all text drawing in Mac OS X, and is thus available on that platform only. ATSUI allows fine control over layout features, provides advanced multilingual text-processing services, and supports high-end typography. For details see http://developer.apple.com/documentation/Carbon/Conceptual/ATSUI_Concepts/.

⁴*International Components for Unicode*. ICU is a widely portable set of C/C++ and Java libraries providing Unicode and globalization support for software applications. ICU ensure that applications give the same results on all platforms and between C/C++ and Java software, see <http://www.icu-project.org/>.

⁵Graphite is a project to provide rendering capabilities for complex non-Roman writing systems. Graphite runs on various computer platforms and allows the creation of “smart fonts” which support displaying in writing systems with various complex behaviors. Details are available at <http://scripts.sil.org/RenderingGraphite>.

LaTeX integration for XeTeX is for a large part the work of Will Robertson.¹ Although XeTeX accepts Unicode input and supports OpenType fonts, XeTeX's interfaces with OpenType fonts is rather low-level. For instance, font features, such as using lowercase numbers instead of uppercase numbers, are activated with hard to remember strings such as `+onum`. Will Robertson's `fontspec` package provides a more readable and easy to use interface to such things with keyval-type options, such as `"Numbers=Lowercase"` for the above example. If you want to use a new OpenType (or TrueType) font, you no longer need to mess around with extra files for font metrics and font definitions. It is sufficient to declare your new font with the command `\setmainfont` in the preamble to select that font as the main document font.

By default, LaTeX's NFSS mechanism only deals well with "macroscopic" font variations, such as weight, shape and size. `fontspec` extends LaTeX's font handling by providing support for "font features", which allow the user at any point in the document to vary a broad range of typographical details by using different font instances.

2.1.1 A brief history

- April 2004: XeTeX 0.3 was released to the TeX community (on Mac OS X only) and offered:
 - integrated Unicode support
 - access to all fonts installed on the computer
 - AAT (Apple Advanced Typography) for typographic features
 - *Quicktime* for graphics support
- February 2005 : XeTeX 0.9 was released with as features:
 - OpenType support
 - compatibility with more important LaTeX packages
- April 2006 (BachTeX): XeTeX for Linux was released (first public announcement of the availability of XeTeX)
- June 2006: Akira Kakuto announces the availability of XeTeX on MS Windows
- February 2007: TeXLive 2007 contains XeTeX 0.996 for all supported binary platforms
- September 2007: XeTeX 0.997 available with MikTeX 2.7 (beta)
- Fall 2008: TeXLive 2008 contains XeTeX 0.999 for all supported binary platforms
- Summer 2009: TeXLive 2009 contains XeTeX 0.999.5 for all supported binary platforms

2.1.2 XeTeX: basic principles

- based on e-TeX's typesetting engine
- includes `TeX--XeT` (commands `\beginL`, `\endL`, `\beginR` and `\endR` activated with `\TeXXeTstate=1`) for bi-directional typesetting (Arabic, Hebrew, etc.)
- Unicode encoding (UTF-8 or UTF-16) used by default
- most LaTeX extensions (e.g., `graphics`, `xcolor`, `geometry`, `crop`, `hyperref`, `pgf`) now automatically detect the presence of the XeTeX engine and are compatible with it
- directly uses OpenType, TrueType and PostScript fonts installed on the system without the need to create TeX-specific files (`.tfm`, `.vf`, `.fd`, etc.)

¹See <http://tug.org/interviews/interview-files/will-robertson.html> for an interview with Will Robertson.

- provides access to OpenType features (ligatures, swash, glyph alternatives, dynamic attachment of accents, etc.)
- thanks to Unicode provides access to characters in extended alphabetic (Latin, Cyrillic, Greek, Arabic, Devanagari, etc.) and complex scripts.
- allows the concurrent use of multiple scripts in a single document thus making processing multi-lingual texts much simpler

Xe_{La}TeX's direct use of Unicode characters as input and of OpenType Unicode-encoded fonts makes pre-processors or complex macros for handling composite characters or complex scripts mostly unnecessary. As an example let us consider the way TeX and Xe_{La}TeX handle some input

TeX input	Xe _{La} TeX input	typeset output	notes
\'{a} \e \^{o}	á è ô	á è ô	typical accents
\c{c} \AA	ç Å	ç Å	composed characters
d\v{z}abe {\dj}ak	džabe đak	džabe đak	more composed characters
---	\char"2014	—	specific ligature in TeX fonts
\\$alpha\\$	\char"1D6FC	α	mathematical symbol (plane 1)
{\dn acchaa}	अच्छा	अच्छा	TeX needs <i>ad hoc</i> preprocessor

2.2 Xe_{La}TeX: typesetting with glyphs, characters and fonts

Xe_{La}TeX delegates the rendering of Unicode characters to the freetype library¹ and uses the font configuration library fontconfig² for accessing font files (other than TeX-specific fonts). The fontconfig library lets you configure, customize and manage fonts for all applications which need to access fonts present on your computing system.

2.2.1 Accessing font with fontconfig

The information concerning fonts is stored in XML format³ and you, as user, should specify where your OpenType fonts live in the file `$HOME/.fonts.conf`, as in the following example of such a file.

```
<?xml version="1.0"?>
<!DOCTYPE fontconfig SYSTEM "fonts.dtd">
<!-- /etc/fonts/fonts.conf file to configure system font access -->
<fontconfig>
<dir>/home/goossens/texlive/2007/texmf-update/fonts/opentype</dir>
<dir>/home/goossens/texlive/2007/texmf-commercial/fonts/opentype</dir>
<dir>/home/goossens/texlive/2007/texmf-dist/fonts/opentype</dir>
</fontconfig>
```

On Microsoft Windows, when running MikTeX, the file `fonts.conf` contains a line to include the file `localfonts.conf`. Both these files live in the directory

```
c:\Documents and Settings\All Users\Application Data\MiKTeX\2.7\fontconfig\config
```

¹See <http://sourceforge.net/projects/freetype/>.

²See <http://fontconfig.org/wiki/>. You need at least fontconfig version 2.4 for Xe_{La}TeX to function correctly.

³These files use a syntax defined by a grammar specified as a DTD (`/etc/fonts/fonts.dtd`). The system-wide configuration file lives in `/etc/fonts/fonts.conf`.

The file `localfonts.conf` has the following content.

```
<?xml version="1.0"?>
<fontconfig>
<dir>C:\WINNT\Fonts</dir>
<dir>C:\Program Files\MiKTeX 2.7\fonts/type1</dir>
<dir>C:\Program Files\MiKTeX 2.7\fonts/opentype</dir>
<dir>c:\TeXlive2007\texmf-dist\fonts\opentype</dir>
<dir>c:\TeXlive2007\texmf-update\fonts\opentype</dir>
<dir>c:\TeXlive2007\texmf-commercial\fonts\opentype</dir>
</fontconfig>
```

Note that MiKTeX includes by default Microsoft Window's (`\WINNT\Fonts`), as well as its own standard font directories. We added three other ones from the T_EXLive trees (as in the example above).

The `fontconfig` library comes with three programs, two for providing information about the font files declared (i.e., findable by `fontconfig`) on your system (`fc-match` and `fc-list`), and one (`fc-cache`) for (re)generating a font cache of all fonts (a `fc-cache` command should be issued each time a new font is installed or deleted).

```
> fc-list --help
usage: fc-list [-vV?] [--verbose] [--version] [--help] [pattern] element ...
List fonts matching [pattern]
-v, --verbose      display status information while busy
-V, --version      display font config version and exit
-?, --help         display this help and exit

> fc-match --help
usage: fc-match [-svV?] [--sort] [--verbose] [--version] [--help] [pattern]
List fonts matching [pattern]
-s, --sort         display sorted list of matches
-v, --verbose      display entire font pattern
-V, --version      display font config version and exit
-?, --help         display this help and exit

> fc-cache --help
usage: fc-cache [-frsvV?] [--force|--really-force] [--system-only] [--verbose] [--version] [--help] [dirs]
Build font information caches in [dirs]
(all directories in font configuration by default).
-f, --force        scan directories with apparently valid caches
-r, --really-force erase all existing caches, then rescan
-s, --system-only  scan system-wide directories only
-v, --verbose      display status information while busy
-V, --version      display font config version and exit
-?, --help         display this help and exit

> fc-list 'Minion Pro'
Minion Pro,Minion Pro Subh:style=Italic Subhead,Italic
Minion Pro:style=Bold Italic
Minion Pro,Minion Pro SmBd Cond Capt:style=Semibold Cond Caption,Regular
Minion Pro,Minion Pro Cond Disp:style=Bold Cond Display,Bold
Minion Pro,Minion Pro Disp:style=Display,Regular
Minion Pro,Minion Pro SmBd Subh:style=Semibold Italic Subhead,Italic
Minion Pro,Minion Pro SmBd Cond Capt:style=Semibold Cond Italic Caption,Italic
Minion Pro,Minion Pro Capt:style=Bold Caption,Bold
Minion Pro,Minion Pro Cond Subh:style=Bold Cond Italic Subhead,Bold Italic
Minion Pro,Minion Pro SmBd:style=Semibold,Regular
Minion Pro,Minion Pro Cond Disp:style=Bold Cond Italic Display,Bold Italic
Minion Pro:style=Regular
...
Many more lines
...
Minion Pro:style=Bold
Minion Pro,Minion Pro Cond:style=Bold Cond,Bold
Minion Pro,Minion Pro Cond:style=Bold Cond Italic,Bold Italic
Minion Pro,Minion Pro SmBd:style=Semibold Italic,Italic
Minion Pro,Minion Pro Disp:style=Italic Display,Italic
```

2.2.2 Specifying 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 Xe_{La}TeX 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:

- Xe_{La}TeX uses operating system applications program interfaces that expect UTF-16 encoded streams, so working with this encoding form avoids the need for conversion at this interface.
- Many of standard TeX’s internal tables are implemented as 256-element arrays indexed by character code. In Xe_{La}TeX these arrays have been enlarged to 65,536 elements each to allow them to be indexed by UTF-16 code values.¹
- 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 are 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 do not need to be included in automatic hyphenation patterns.

In view of these factors, Xe_{La}TeX 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.

Xe_{La}TeX uses Unicode’s 16-bit UTF-16 encoding

- characters encoded in 16 bits
 - uses Unicode’s UTF-16 encoding
 - exception: a few ancient differently-encoded fonts
- extension of TeX primitives
 - `\char`, `\chardef` accept numbers up to 65536
 - four-digit notation using the syntax `^^^abcd`
`\char"5609^^^6167 = 嘉慧`
- Unicode characters in the upper (> 0) planes
 - use of surrogates (standard UTF-16)
 - all right for typesetting

¹In principle, using full 32-bit wide arrays would be possible but they would make for extremely large arrays and have a very large memory footprint. Some kind of sparse array implementation would be necessary, but this requires significant additional development and testing, and might impact performance of key inner-loop parts of the TeX system. Therefore the more pragmatic 16-bit approach has been adopted.

- does not allow text manipulation in the input stream on the level of the individual character
- increased size for internal code tables for `\catcode`, `\lccode`, `\uccode`, `\sfcode`
 - “XeTeX plain” initialises its tables with Unicode code points
 - `\lowercase{DŽIN}` `džin`
 - `\uppercase{Esi eyama klɔ míáfe nuvɔwo ɖa vɔ la}`
`ESI EYAMA KLɔ MÍÁFE NUVɔWO ɖA Vɔ LA`
 - `\catcode`\𐌆=\active \def𐌆{...}`

XeTeX’s default input encoding is Unicode (UTF-8 or UTF-16). XeTeX automatically detects the encoding used in the input file. If a non-Unicode encoding is used, it has to be specified with a `\XeTeXinputencoding` command (see page 41). Such historical encodings are handled with the ICU conversion routines.

2.2.3 Hyphenation

At the moment XeTeX reuses T_EX’s hyphenation patterns by adding an extra Unicode layer provided by language-specific intermediate files in the `xu-hyphen` directory¹. An example of such a file (`xu-frhyph.tex` which handles the French patterns) follows.

```

##### xu-frhyph.tex (Wrapper for XeTeX to read frhyph.tex)
\begingroup
\expandafter\ifx\curname XeTeXrevision\endcsname\relax
\else
  % frhyph.tex uses ^^xx for T1 characters
  % redefine them to access the required Unicode characters
  % (only \oe{} actually matters here!)
  \input xu-t1.tex
\fi
\input frhyph.tex
\endgroup

```

It is seen that `xu-frhyph.tex` first loads the generic file `xu-t1.tex`, which makes the letters in the T1-encoded hyphenation pattern files active to map them onto their Unicode equivalents. Part of the contents of that files follows.

```

##### xu-t1.tex #####
% make T1 letters \active and map them to Unicode character codes
% (for use when loading hyphenation patterns that use ^^xx notation
% to represent characters in T1 font encoding, or literal 8-bit
% bytes if read using \XeTeXinputencoding "bytes")
\catcode`\`=12 % ensure " isn't active or otherwise "weird"
\catcode`\^=7 % ensure ^ is the proper catcode for hex notation
%
\catcode"B0=\active \def^^b0{^^^0159} % rcaron
...
\catcode"DF=\active \def^^df{SS} % SS
\catcode"F7=\active \def^^f7{^^^0153} % oe
\catcode"D7=\active \def^^d7{^^^0152} % OE

```

¹With T_EXLive this directory is at `texmf-dist/tex/generic/xu-hyphen`.

```
% we don't handle the non-letter codes in the control range
% but we'd better handle dotless-i (for Turkish)
\catcode"19=\active \def^^19{^^^0131} % dotlessi
```

For languages that do not use the Latin alphabet other similar redefinitions are made in the intermediate files. On top of that fully UTF-8encoded files exist for ancient, monotonic and polytonic modern Greek and for Coptic.

To hyphenate words correctly hyphenation patterns have also been extended to 16 bits. As described previously, an interface between 8-bit pattern files and X_YTeX's 16-bit variants exists. For pure Unicode pattern files are simple Unicode data, without need of commands or active characters, as the following examples show.

```
% hyphenate before and after independent vowel
1ᄁ1
1ᄁ1
1ᄁ1
% hyphenate following an independent vowel but never before
2ᄁ1
2ᄁ1
```

2.2.4 Font management: the basics

X_YTeX can use all modern font formats (PostScript Type 1, TrueType, OpenType) and gives access to all fonts on your computer. Moreover, X_YTeX lets you still use T_EX-specific font files, such as `tfm`. The latter are useful for math fonts or for non-Unicode encoded input files.

X_YTeX extends T_EX's `\font` command (as explained later). In particular, you can specify the actual name of a font, rather than its somewhat artificial 8-character equivalent in the *Fontname* scheme.¹ Examples are

- `\font\rm="Adobe Caslon Pro" at 14pt \rm Bonjour GUT2007 !`
Bonjour GUT2007 !
- `\font\it="Trebuchet MS" at 14pt \it Bonjour GUT2007 !`
Bonjour GUT2007 !
- `\font\ch="Viva Std" at 14pt \ch Bonjour GUT2007 !`
Bonjour GUT2007 !

A PDF post-processor (by default `xdvipdfmx` on Linux) can use the three font formats mentioned above. `xdvipdfmx` has access to all fonts usable by `xetex`, i.e., those in font directories declared to `fontconfig` or in T_EX's `texmf` font trees of (this is in analogy to `dvips`). On the other hand, `xdvipdfmx` has no support for bitmap fonts and limited `xdvipdfmx` generates PDF by default. It only includes the characters of a font that are actually referenced into the PDF file. `xdvipdfmx` can generate an intermediate “extended DVI” format (`.xdv`). This intermediate format can be useful when `xetex` encounters an error and does not generate a PDF file. In that case you can use the following two-step process to investigate the problem (note the use of the “verbosity” switch `-vv`).

```
> xelatex -no-pdf mydocument
> xdvipdfmx -vv -E mydocument.xdv
```

¹*Fontname* is maintained by Karl Berry. Its documentation is available as an electronic document on CTAN at: `info/fontname`.

2.2.5 Font mappings using TECKit

TECKit (currently version 2.2, see <http://scripts.sil.org/TECKit>) is a low-level toolkit intended to be used by other applications that need to perform encoding conversions (e.g., when importing legacy data into a Unicode-based application). The primary component of the TECKit package is therefore a library that performs conversions; this is the “TECKit engine”. The engine relies on mapping tables in a specific binary format (for which documentation is available); there is a compiler that creates such tables from a human-readable mapping description (a simple text file).

Widely-used T_EX keyboarding conventions such as `\' {e} → “é”` or `\pounds → £` are implemented via T_EX 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 associated with standard T_EX fonts; these include `--- → —` (em-dash), `?` → “¿”` (Spanish inverted “?”), 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 T_EX fonts. But as these conventions are peculiar to the T_EX 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, X_YT_EX 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.

While TECKit was primarily designed to convert between legacy byte encodings and Unicode, it can also be used to perform transformations on a Unicode text stream, using the same mapping language and text conversion library. The following shows the file `tex-text.map` (in fact its binary equivalent `tex-text.tec`, which usually lives in the `texmf` tree in subdirectory `texmf/fonts/misc/xetex/fontmapping/`), which provides support for normal T_EX conventions.

```

; TECKit mapping for TeX input conventions <-> Unicode characters
; used with XeTeX to emulate Knuthian ligatures

; Copyright 2006 SIL International.
; You may freely use, modify and/or distribute this file.

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

```

When associated with a standard Unicode-compliant font in X_YT_EX, this has the effect of imple-

menting the legacy TeX conventions for dashes and quotes, as shown in the next example, without requiring any TeX-specific features in the smart fonts themselves.

Exa.
2-2-1

```
!Typing "quotes"---and dashes---the TeX way!
!Typing "quotes"—and dashes—the TeX way!
```

```
\font\TestA="Times New Roman" at 9pt
\TestA !`Typing "quotes"(1--2)---and
  ``dashes' '---the \TeX\ way!\par
\bigskip
\font\TestB="Times New Roman:
  mapping=tex-text" at 9pt
\TestB !`Typing "quotes"(1--2)---and
  ``dashes' '---the \TeX\ way!\par
```

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. The following example shows how to typeset a single fragment of input text in two scripts by giving different font specifications, one of which includes a transliteration mapping (in this case the mapping file `cyr-lat-iso9.tex` must be findable by XeTeX).

Exa.
2-2-2

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

```
\def\SampleText{Unicode \\  
  это уникальный  
  код для любого символа,\\  
  независимо от платформы,\\  
  независимо от программы,\\  
  независимо от языка.\par}
```

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

```
\font\gen="Gentium" at 9pt
\centering
\gen\SampleText
\bigskip
\font\gentrans="Gentium:mapping=cyr-lat-iso9"
  at 9pt \gentrans
\SampleText
```

2.2.6 Line breaks and justification

Some languages do not use spaces between words in the input file, so the line breaks must be generated when typesetting the text.

- TeX normally breaks line at a point where there is “glue” associated to an inter-word space
- Chinese, Japanese, Thai, etc. do not leave spaces between words
- โดยพื้นฐานแล้ว, คอมพิวเตอร์จะเกี่ยวข้องกับเรื่องของตัวเลข. คอมพิวเตอร์จัดเก็บตัวอักษรและอักขระอื่นๆ โดยการกำหนดหมายเลขให้สำหรับแต่ละตัว. ก่อนหน้าที่ Unicode จะถูกสร้างขึ้น, ได้มีระบบ encoding อยู่หลายร้อยระบบสำหรับการกำหนดหมายเลขเหล่านี้.

The linebreaking model implemented in the ICU library is used with: `\XeTeXlinebreaklocale "th"`

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

Line justification of a text without spaces, including line breaking is a non-trivial task. One solution is ragged typesetting (i.e., no text alignment to the right (or left) margin).

- 基本上，计算机只是处理数字。它们指定一个数字，来储存字母或其他字符。在创造Unicode之前，有数百种指定这些数字的编码系统。没有一个编码可以包含足够的字符：

Alternatively one can use the command `\XeTeXlinebreakskip`, which lets you introduce glue at potential beak points.

- 基本上，计算机只是处理数字。它们指定一个数字，来储存字母或其他字符。在创造Unicode之前，有数百种指定这些数字的编码系统。没有一个编码可以包含足够的字符：

2.2.7 Unicode 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. Currently, XeTeX supports two such rendering systems: ATSUI on Mac OS X, and ICU on other systems.

2.2.8 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. Therefore, the system was extended by incorporating the OpenType layout engine from ICU.¹ 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. 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.

¹In addition to the actual layout engine, XeTeX uses ICU’s implementation of Unicode’s BiDI (bi-directional) algorithm.

When the T_ƎX source calls for a particular font, X_ƎT_ƎX looks for specific layout tables within the font (e.g., GSUB for OpenType) to determine which layout engine to use, and instantiates either an ATSUI style or an ICU LayoutEngine as appropriate (for a font that supports both layout technologies, X_ƎT_ƎX currently chooses the OpenType engine by default, but users can explicitly specify which one to use). The difference in the implementation of the two technologies is, however, entirely hidden from the main T_ƎX program, which simply deals with “word nodes”, forming them into paragraphs and pages once they have been measured by the appropriate smart-font engine.

X_ƎT_ƎX optimally exploits the Unicode characteristics present in OpenType fonts. Therefore, X_ƎT_ƎX differs rather drastically from T_ƎX’s traditional model, characterized by:

- T_ƎX’s fundamental typesetting unit is a code point of a given character in a particular font, where T_ƎX assumes that the dimensions of such a character are known and invariable
- ligatures are handled by a character substitution mechanism
- a paragraph is constructed from a sequence of *character* nodes, which are placed with great precision, interspersed with nodes of *glue*.

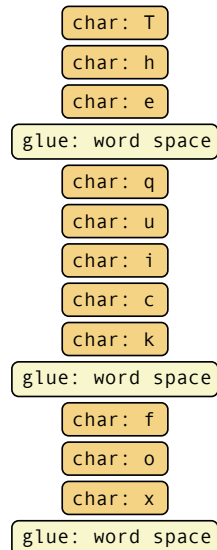
This is not optimal for Unicode, where a character might not correspond to a single known glyph. Indeed, many scripts require contextual selection of glyphs (e.g., Arabic, Devanagari), so that characters must be measured in context rather than in isolation.

X_ƎT_ƎX’s approach is the following:

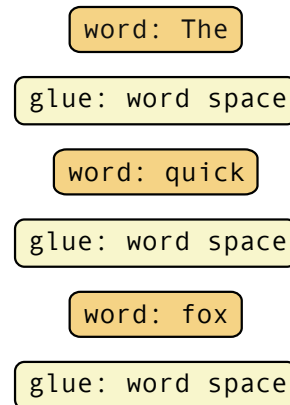
- the typesetting process collects runs of characters (words) whose widths are obtained via the API to the system libraries (e.g., ICU) to determine the widths,
- a X_ƎT_ƎX paragraph is a sequence of *word* nodes separated by *glue*.

Thus X_ƎT_ƎX’s typesetting engine places words rather than glyphs, the latter being drawn by the font rendering engine. The following scheme illustrates this distinction between the T_ƎX and X_ƎT_ƎX engines.

T_ƎX : nodes in a paragraph



X_ƎT_ƎX : nodes in a paragraph



Depending on the tables present in a given font, X_ƎT_ƎX will use ATSUI (the equivalent of ICU on Mac OS X) or ICU and localizes the requested font with the application fontconfig. Thus, the typesetting process is completely independent of the underlying font technology (only the low-level layout engine,

which needs to determine the dimensions of the characters, has to know. Therefore a given source file can refer at the same time to OpenType, AAT, and even TeX fonts).

By default X_YTeX uses the xdvipdmtx output engine, which uses the freetype library (www.freetype.org/) for rendering the images of the glyphs with great precision.

2.2.9 X_YTeX's 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.

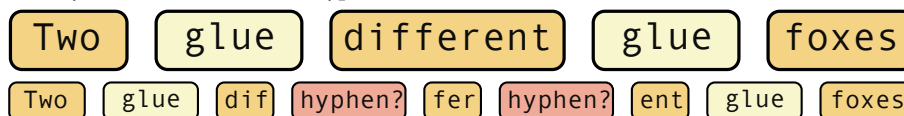
Explicit discretionary hyphens may be included in TeX input, and these continue to work in X_YTeX, as they become “discretionary break” nodes in the list of items making up the paragraph. The word fragments on either side, then, would become separate nodes in the list, and a line-break can occur between 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 X_YTeX, a user reported that OpenType ligatures in certain words such as *different* would intermittently fail (appearing as *differ^{ent}*, 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.

- a paragraph is built from a list of *word boxes*
 - these boxes are treated as indivisible units in the *token lists*
 - TeX can remain unaware of low-level details
- when an acceptable linebreak cannot be found the algorithm tries to hyphenate words
 - extract the characters from the *word nodes*
 - find break points using TeX's hyphenation algorithm
 - repackage words as word fragments and discretionary hyphenation nodes

- modify the node list to allow hyphenation of words



- problem : the unused hyphenation points break rendering

Two glue dif fer -
ent glue foxes

*Two differ-
ent foxes*

- one has to re-merge word nodes after choosing breaks

Two glue differ-
ent glue foxes

*Two differ-
ent foxes*

2.2.10 Running xetex

As explained in Section 2.1.2 X_YT_EX is a development of e-_TE_X and it builds on Karl Berry's kpathsea library for path searching as implemented in the Web2C version of _TE_X.¹ The xetex command thus offers essentially the same options (type `xetex --help` to get a full list) as the tex command (e.g., the version distributed with _TE_XLive). The more important additional ones are:

- etex enable the e-_TE_X extensions
- no-pdf generate XDV (extended DVI) output rather than PDF (see also page 27)
- output-driver=CMD use CMD as the XDV-to-PDF driver instead of xdvipdfmx, the default driver used by xetex

2.3 Supplementary commands introduced by X_YT_EX

X_YT_EX offers a few additional features, most of which are available with the help of new commands or via the higher level _ET_EX interface of Will Robertson's fontspec package.

X_YT_EX extends _TE_X's basic `\font` command with additional options to address the rich set of features available in OpenType (and AAT) fonts, as follows.

```
\font\myname="[fontname]{font-options}:{font-features}"{TEX font-features}
```

The only mandatory part of this construct is *fontname*, the actual name of the font (as encoded in the .ttf or .otf files, e.g., TeX Gyre Schola).

The xdvipdfmx driver can also use fonts that are not installed in the operating system. Such fonts should have their name specified in square brackets. The full path can be specified in the font declaration, as follows,

```
\font\myname="/mydir/myfontfile"
```

Alternatively, the current directory and the texmf trees can be searched for locating the given filename, e.g., the following will select a Latin Modern font in the user's _TE_X hierarchy.

```
\font\myname="[lmroman10-regular]"
```

¹The Web2C implementation of the _TE_X family of programs is a translation of the original WEB sources of these program into the C programming language to allow easy compilation on all present-day computer systems. A detailed description is available from its Web page (<http://www.tug.org/web2c/>) where you can find also the kpathsea manual. Currently Web2C is part of _TE_XLive.

The argument *font options* can only be used when the font is selected through the operating system (i.e., without square brackets), and may be any concatenation of the following:

```

/B Use the bold version of the selected font.
/I Use the italic version of the selected font.
/BI Use the bold italic version of the selected font.
/IB Same as /BI.
/S=x Use the version of the selected font corresponding to the optical size x pt.
/AAT Explicitly use the ATSUI renderer (Mac OS X only).
/ICU Explicitly use the ICU OpenType renderer (only useful on Mac OS X).
```

The argument *font-features* is a comma or semi-colon separated list activating or deactivating various AAT or OpenType font features, which will vary by font. The X_YT_EX distribution contains the documentation file `opentype-info.tex` which lists all supported features available for the various scripts and languages in the specified OpenType font.¹

OpenType font features are chosen by specifying their standard tags names,² separated by a comma or a semicolon, and prepended with a + to turn them on, or - to turn them off.

Bold italic Minion Pro
SMALL CAPS BOLD ITALIC MINION PRO

```

\font\wbi="Minion Pro/BI" at 12pt
\wbi Bold italic Minion Pro\par
\font\wbisc="Minion Pro/BI:+smcp" at 12pt
\wbisc Small caps bold italic Minion Pro
```

Exa.
2-3-1

X_YT_EX offers a series of features that are available for any font, namely

mapping= Specifies the mapping for the given font. For example, `mapping=te`
`xt` enables “classical” T_EX mappings such as the sequence “---” being turned into the proper
 typographical glyph “—”, etc.

color=RRGGBB[TT] Specifies the color for the given font as three pairs of hexadecimal RGB values.
 An optional argument lets you specify a transparency value.

letterspace=x A space of x/S is added between words (S is the font size).

Depending on the script and language chosen a certain number of OpenType features, when avail-
 able, will be activated by default.

Script and language are chosen as follows:

script=<script tag> selects the font script,

language=<lang tag> selects the font language.

Script (alphabet) tags are four-letter codes,³ while language tags are three-letter codes.⁴

2.3.1 Specifying languages and scripts

Certain characters have a different presentation depending on the language in which they are used.
 Below we show how identical input texts are rendered with identical fonts first in the default language

¹A similar file, `aat-info.tex`, exists for displaying the characteristics of an AAT font.

²See <http://www.microsoft.com/typography/otspec/featuretags.htm> for a list of available registered features.

³See <http://www.microsoft.com/typography/otspec/scripttags.htm>.

⁴See <http://www.microsoft.com/typography/otspec/languagegetags.htm>.

(left) and then in Vietnamese, respectively, Turkish (right).

```
\font\Doulos="Doulos SIL"           \font\DoulosViet="Doulos SIL:language=VIT"
```

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

```
\font\Minion="Minion Pro"           \font\MinionTrk="Minion Pro:language=TRK"
```

gelen firmaları ... tarafından gelen firmaları ... tarafından

Moreover, certain languages need a language-specific rendering procedure to draw the form of the letters, as the following examples of Arabic and Devanagari show.

- `\font\x="Code2000:script=arab" \x` يبرعلا → العربي
- `\font\x="Code2000:script=deva" \x` हन्दि → हिन्दी

2.3.2 Specifying optional features

The font declaration can refer to one or more optional features.

- `\font\x="Minion Pro" \x` Hello TUG2008! 0123456789
Hello TUG2008! 0123456789
- `\font\x="Minion Pro:+smcp" \x`
HELLO TUG2008! 0123456789
- `\font\x="Minion Pro Italic:+onum" \x`
Hello TUG2008! 0123456789
- `\font\x="Minion Pro Italic:+swsh,+zero" \x`
Hello TUG2008! 0123456789

Certain fonts come in a several optical sizes, so that the image of the character is optimized to the typeset size used.

- Minion Pro typeset at 7pt, at 10pt, at 18pt, and at 24pt

seven ten **eighteen twenty four**

One can force a given optical size as shown with the following texts which are all typeset at 16pt, but which use the optical size specified with the `/S=` specifier.

Minion Pro/S=7	Minion Pro Caption
Minion Pro/S=10	Minion Pro Text
Minion Pro/S=18	Minion Pro Subhead
Minion Pro/S=24	Minion Pro Display

2.3.3 Support for pseudo-features

Sometimes it can be useful to “fake” some features by emulating them when they are not natively available in a given font. Examples are slanting (in the absence of a genuine Italic variant) or extending the width of a font (when wider or condensed variants do not exist). These effects can be achieved with the `slant` and `extend` pseudo-features, as the following example shows.

Charis SIL normal

Charis SIL slanted

Charis SIL extended

Charis SIL condensed, slanted

Charis SIL condensed, anti-slanted

```
\font\x="Charis SIL" at 12 pt
\x Charis SIL normal\l[1mm]
\font\x="Charis SIL:slant=0.2" at 12 pt
\x Charis SIL slanted\l[1mm]
\font\x="Charis SIL:extend=1.5" at 12 pt
\x Charis SIL extended\l[1mm]
\font\x="Charis SIL:slant=0.2;extend=0.8" at 12 pt
\x Charis SIL condensed, slanted\l[1mm]
\font\x="Charis SIL:slant=-0.2;extend=0.8" at 12 pt
\x Charis SIL condensed, anti-slanted
```

Exa.
2-3-2

2.3.4 Commands extracting information from OpenType fonts

XeTeX provides new commands to extract information from font files.

`\XeTeXuseglyphmetrics`

A counter which specifies whether the height and depth of characters must be taken into account in the typesetting process (>0, the default), or whether a single height and depth for all characters is used (<1).

m M g G

m M g G

```
\font\minion="Minion Pro" at 12pt\minion
\XeTeXuseglyphmetrics=0 \fbox{m}\fbox{M}\fbox{g}\fbox{G}
\par\medskip
\XeTeXuseglyphmetrics=1 \fbox{m}\fbox{M}\fbox{g}\fbox{G}
```

Exa.
2-3-3

`\XeTeXglyph{Glyph slot}`

Inserts the glyph in *slot* of the current font (font specific, i.e., this command will give different output for different fonts).

`\XeTeXglyphindex"glyphname"`

This command, that must be followed by a space or `\relax`, returns the *glyph slot* corresponding to the (possibly font specific) *glyphname* in the currently selected font.


```
\XeTeXcharglyph{charcode}
```

This command returns the default glyph number of character *charcode* in the current font (the value of zero is returned if the character is absent from the font).

Exa.
2-3-4

The glyph slot in Minion Pro for the copyright symbol is: 170 (using the font-specific glyph name) or 170 (using the unicode character slot).

This glyph may be typeset with the font-specific glyph slot printed above [®], or directly by storing the slot number in a counter, as follows: ©. The Unicode code can also be used directly to address the character slot, as follows: © (TeX syntax) or © (L^AT_EX syntax).

```
\font\minion="Minion Pro"\minion
\raggedright
The glyph slot in Minion Pro for the copyright symbol is:
\the\xeTeXglyphindex"copyright" \space (using the font-specific glyph
name) or \the\xeTeXcharglyph"00A9 \space (using the unicode character
slot).

\newcounter{Cslot}
\setcounter{Cslot}{\the\xeTeXglyphindex"copyright"}
\medskip
This glyph may be typeset with the font-specific glyph slot printed
above \XeTeXglyph170, or directly by storing the slot number in a
counter, as follows: \XeTeXglyph\value{Cslot}. The Unicode code can
also be used directly to address the character slot, as follows:
\char"00A9 \space (\TeX{} syntax) or \symbol{"00A9} (\LaTeX{} syntax).
```

```
\XeTeXfonttype{font}
```

Returns the number corresponding to the renderer which is used for *font*:

- 0 for TeX (standard TeX-based .tfm font);
- 1 for ATSUI (usually an AAT font);
- 2 for ICU (an OpenType font);
- 3 for Graphite.

Exa.
2-3-5

"[cmtt10]" is rendered by ICU.
 "LMRoman10 Regular" is rendered by ICU.
 "[lmsans10-bold]" is rendered by ICU.
 "Charis SIL" is rendered by ICU.
 "Charis SIL/AAT" is rendered by ICU.

```
\usepackage{ifthen}
\newcounter{Cfont}
\newcommand\whattype[1]{%
  \texttt{\fontname#1} is rendered by
  \setcounter{Cfont}{\XeTeXfonttype#1}
  \ifthenelse{\value{Cfont}=0}{\TeX}{%
    \ifthenelse{\value{Cfont}=1}{ATSUI}{%
      \ifthenelse{\value{Cfont}=2}{ICU}{%
        \ifthenelse{\value{Cfont}=3}{Graphite}%
        {\typeout{Renderer number not known}}}}}%
.\par}

\font\fa="cmtt10"
\font\fb="LMRoman10 Regular"
\font\fc="[lmsans10-bold]"
\font\fd="Charis SIL"
\font\fe="Charis SIL/AAT"
\whattype\fa\whattype\fb
\whattype\fc\whattype\fd\whattype\fe
```

```
\XeTeXOTcountscripts{Font}
```

Returns the number of scripts present in a font.

The number of scripts in Minion Pro is 4.
 The number of scripts in Charis SIL is 2.
 The number of scripts in Arial Unicode MS is 8.
 The number of scripts in Code2000 is 21.

```
\newcommand{\NumScripts}[1]{%
  \font\testfont="#1"\testfont
  The number of scripts in #1 is
  \the\xeTeXOTcountscripts\testfont.}
\NumScripts{Minion Pro}\par
\NumScripts{Charis SIL}\par
\NumScripts{Arial Unicode MS}\par
\NumScripts{Code2000}
```

Exa.
2-3-6

```
\XeTeXOTscripttag{Font}{n}
```

Expands to a counter corresponding to script tag *n* in the font.

```
\XeTeXOTcountlanguages{Font}{ScriptTag}
```

Expands to counter corresponding to the number of languages supported by the given script in the font.

```
\XeTeXOTlanguagetag{Font}{ScriptTag}{n}
```

Expands to a counter corresponding to language tag *n* in the given script of the font.

```
\XeTeXOTcountfeatures{Font}{ScriptTag}{LanguageTag}
```

Expands to a counter corresponding to the number of features for the given script and language tags of the font.

Type (Class)	Meaning	Example	Type (Class)	Meaning	Example
<code>\mathord</code> (0)	Ordinary	/	<code>\mathopen</code> (4)	Opening	(
<code>\mathop</code> (1)	Large operator	<code>\int</code>	<code>\mathclose</code> (5)	Closing)
<code>\mathbin</code> (2)	Binary operation	+	<code>\mathpunct</code> (6)	Punctuation	,
<code>\mathrel</code> (3)	Relation	=	<code>\mathalpha</code> (7)	Alphabet character	A

Table 2.1: Mathematics symbol types

```
\XeTeXOTfeaturetag{Font}{ScriptTag}{LanguageTag}{n}
```

Expands to a counter corresponding to feature tag n for the given script and language tags in the font.

A file `OpenType-info.tex` that is available with the XeTeX distribution uses all the commands to list the features for all languages and scripts supported by a given OpenType font.

2.3.5 Maths fonts

To handle maths parameters more easily XeTeX adds a series of new primitives to standard TeX. In the description of these supplementary commands that follows, Fam is a number (0–255) representing the font to use in maths and $MathType$ is an integer in the range 0–7 (Table 2.1) defining the nature (*class* in TeX language, see [4, p. 154]) of the math symbol, i.e., whether it is a binary operator, a relation, etc. (La)TeX needs this information to leave the correct amount of space around the symbol when it is used in a formula (see [5, Section 8.9] for more details).

```
\XeTeXmathcode{char slot} [=] {MathType} {Fam} {GlyphSlot}
```

Defines a maths glyph accessible via an input character. Note that the input takes *three* arguments unlike TeX’s `\mathcode`.

```
\XeTeXmathcodenum{CharSlot} [=] {MathType/Fam/GlyphSlot}
```

Pure extension of `\mathcode` that uses a “bit-packed” single number argument. Can also be used to extract the bit-packed mathcode number of the $CharSlot$ if no assignment is given.

```
\XeTeXmathchardef{cmd} [=] {MathType} {Fam} {GlyphSlot}
```

Defines a maths glyph accessible via a control sequence.

```
\XeTeXdelcode{CharSlot} [=] {Fam} {GlyphSlot}
```

Defines a delimiter glyph accessible via an input character.

```
\XeTeXdelcodenum{CharSlot} [=] {Fam/GlyphSlot}
```

Pure extension of `\delcode` that uses a “bit-packed” single number argument. Can also be used to extract the bit-packed mathcode number of the $CharSlot$ if no assignment is given.

```
\XeTeXdelimiter{MathType} {Fam} {GlyphSlot}
```

Typesets the delimiter in the $GlyphSlot$ in the family specified of either $MathType$ 4 (opening) or 5 (closing).

OpenType Layout features found in Arial Unicode MS:

```

script = 'arab'
language = 'FAR'
features = 'isol' 'init' 'medi' 'fina' 'liga' 'isot' 'isot1' 'fina' 'locl'
language = 'FRD'
features = 'isol' 'init' 'medi' 'fina' 'liga' 'isot' 'init' 'medi' 'fina'
'locl'
language = <default>
features = 'isol' 'init' 'medi' 'fina' 'liga' 'mark'

script = 'deva'
language = <default>
features = 'nukt' 'akhn' 'rphf' 'blwf' 'half' 'valu' 'pres' 'abvs' 'blws'
'psts' 'hain' 'abvm' 'blwm' 'dist'

script = 'gujr'
language = <default>
features = 'nukt' 'akhn' 'rphf' 'blwf' 'half' 'valu' 'pres' 'abvs' 'blws'
'psts' 'hain' 'abvm' 'blwm' 'dist'

script = 'hani'
language = 'JAN'
features = 'vert'
language = 'KOR'
features = 'locl' 'vert'
language = 'ZHS'
features = 'locl' 'vert'
language = 'ZHT'
features = 'locl' 'vert'
language = <default>
features = 'salt' 'trad' 'smp1' 'vert'

script = 'kana'
language = 'JAN'
features = 'vert'
language = <default>
features = 'vert'

script = 'knda'
language = <default>
features = 'akhn' 'rphf' 'blwf' 'half' 'blws' 'abvs' 'psts' 'hain' 'dist'
'dist'

script = 'taml'
language = <default>
features = 'akhn' 'half' 'abvs' 'psts' 'hain'

```

OpenType Layout features found in Minion Pro:

```

script = 'cyrl'
language = <default>
features = 'aalt' 'c2sc' 'case' 'dlig' 'dnom' 'fina' 'frac' 'hist' 'liga'
'lnum' 'numr' 'ordn' 'ornm' 'pnum' 'salt' 'sinf' 'smcp' 'ss01'
'ss02' 'supr' 'tnum' 'zero' 'cpsp' 'kern' 'size'

script = 'grek'
language = <default>
features = 'aalt' 'c2sc' 'case' 'dlig' 'dnom' 'fina' 'frac' 'hist' 'liga'
'lnum' 'numr' 'ordn' 'ornm' 'pnum' 'salt' 'sinf' 'smcp' 'ss01'
'ss02' 'supr' 'tnum' 'zero' 'cpsp' 'kern' 'size'

script = 'latn'
language = 'AZE'
features = 'aalt' 'c2sc' 'case' 'dlig' 'dnom' 'fina' 'frac' 'hist' 'liga'
'lnum' 'numr' 'ordn' 'ornm' 'pnum' 'salt' 'sinf' 'smcp' 'ss01'
'ss02' 'supr' 'tnum' 'zero' 'cpsp' 'kern' 'size'

language = 'CRT'
features = 'aalt' 'c2sc' 'case' 'dlig' 'dnom' 'fina' 'frac' 'hist' 'liga'
'lnum' 'numr' 'ordn' 'ornm' 'pnum' 'salt' 'sinf' 'smcp' 'ss01'
'ss02' 'supr' 'tnum' 'zero' 'cpsp' 'kern' 'size'

language = 'DEU'
features = 'aalt' 'c2sc' 'case' 'dlig' 'dnom' 'fina' 'frac' 'hist' 'lnum'
'numr' 'ordn' 'ornm' 'pnum' 'salt' 'sinf' 'smcp' 'ss01' 'ss02'
'supr' 'tnum' 'zero' 'cpsp' 'kern' 'size'

language = 'MOL'
features = 'aalt' 'c2sc' 'case' 'dlig' 'dnom' 'fina' 'frac' 'hist' 'liga'
'lnum' 'locl' 'numr' 'ordn' 'ornm' 'pnum' 'salt' 'sinf' 'smcp'
'ss01' 'ss02' 'supr' 'tnum' 'zero' 'cpsp' 'kern' 'size'

language = 'ROM'
features = 'aalt' 'c2sc' 'case' 'dlig' 'dnom' 'fina' 'frac' 'hist' 'liga'
'lnum' 'locl' 'numr' 'ordn' 'ornm' 'pnum' 'salt' 'sinf' 'smcp'
'ss01' 'ss02' 'supr' 'tnum' 'zero' 'cpsp' 'kern' 'size'

language = 'SRB'
features = 'aalt' 'c2sc' 'case' 'dlig' 'dnom' 'fina' 'frac' 'hist' 'liga'
'lnum' 'numr' 'ordn' 'ornm' 'pnum' 'salt' 'sinf' 'smcp' 'ss01'
'ss02' 'supr' 'tnum' 'zero' 'cpsp' 'kern' 'size'

language = 'TRK'
features = 'aalt' 'c2sc' 'case' 'dlig' 'dnom' 'fina' 'frac' 'hist' 'liga'
'lnum' 'numr' 'ordn' 'ornm' 'pnum' 'salt' 'sinf' 'smcp' 'ss01'
'ss02' 'supr' 'tnum' 'zero' 'cpsp' 'kern' 'size'

language = <default>
features = 'aalt' 'c2sc' 'case' 'dlig' 'dnom' 'fina' 'frac' 'hist' 'liga'
'lnum' 'numr' 'ordn' 'ornm' 'pnum' 'salt' 'sinf' 'smcp' 'ss01'
'ss02' 'supr' 'tnum' 'zero' 'cpsp' 'kern' 'size'

```

Figure 2.4: List of features for the scripts and languages supported by the Microsoft Arial and Adobe Minion fonts

```
\XeTeXradical{Fam}{GlyphSlot}
```

Typesets the radical in the *glyph slot* in the family specified.

2.3.5.1 Character classes

The idea behind character classes is to define a boundary where tokens can be added to the input stream without explicit markup. It is primarily intended for automatic alphabet/language font switching.

```
\XeTeXinterchartokenstate
```

Counter. If positive, enables the character classes functionality.

```
\XeTeXcharclass{CharSlot}[=]{ClassNumber}
```

Assigns a class corresponding to *ClassNumber* (range 0–255) to a *CharSlot*. Most characters are class 0 by default. Class 1 is for CJK ideographs, classes 2 and 3 are CJK punctuation. Special case class 256 is ignored; useful for diacritics.

```
\XeTeXinterchartoks{ClassNum1}{ClassNum2}[=]{token list}
```

Defines tokens to be inserted at the interface between *ClassNum1* and *ClassNum2* (in that order).

Exa.
2-3-7

a[A]a

```
\XeTeXinterchartokenstate = 1
\XeTeXcharclass `a 7
\XeTeXcharclass `A 8
\XeTeXinterchartoks 7 8 = {[itshape]}
\XeTeXinterchartoks 8 7 = {[upshape]}
\Large aAa
```

2.3.6 Encodings, linebreaking, etc.

```
\XeTeXversion \XeTeXrevision
```

Expand to a number corresponding to the X_YTeX version, and to a string corresponding to the X_YTeX revision number, respectively.

Exa.
2-3-8

The X_YTeX version is: 0.997

```
\usepackage{xltextra}
The \XeTeX\ version is: \the\XeTeXversion\XeTeXrevision
```

```
\XeTeXinputencoding{CharsetName}
```

Defines the input encoding of the following text.

```
\XeTeXdefaultencoding{CharsetName}
```

Defines the input encoding of subsequent files to be read.

```
\XeTeXdashbreakstate{Integer}
```

Specify whether line breaks after en- and em-dashes are allowed. Off, 0, by default.

```
\XeTeXlinebreaklocale{LocaleID}
```

Defines how to break lines for multilingual text. For instance, to break Chinese text, where the characters are not separated by spaces, one can use the following (see also Example 2-4-7):

```
\XeTeXlinebreaklocale "zh"
```

```
\XeTeXlinebreakskip{Glue}
```

Inter-character linebreak stretch.

```
\XeTeXlinebreakpenalty{Integer}
```

Inter-character linebreak penalty.

```
\XeTeXupwardsmode{Integer}
```

If greater than zero, successive lines of text (and rules, boxes, etc.) will be stacked upwards instead of downwards.

2.3.7 Graphics and pdfT_EX-related commands

This description is incomplete.

```
\XeTeXpicfile{Filename}{Options}
```

Insert an image.

```
\XeTeXpdffile{Filename}{Options}
```

Insert (pages of) a PDF. A simple example of how to include a one-page PDF file follows.

```
\XeTeXpdffile "myfile.pdf"
```

```
\pdfpageheight{Dimension}
```

The height of the PDF page.

```
\pdfpagewidth{Dimension}
```

The width of the PDF page.

```
\pdfsavepos
```

Saves the current location of the page in the typesetting stream.

```
\pdflastxpos
```

Retrieves the horizontal position saved by the above.

```
\pdflastypos
```

Retrieves the vertical position saved by the above.

2.4 fontspec

As explained previously, Jonathan Kew's $X_{\text{E}}\text{TeX}$ lets you easily use all OpenType (and TrueType) fonts available on your computer system with TeX without having to create a whole series of `.tfm`, `.vf`, etc. files. Nevertheless $X_{\text{E}}\text{TeX}$'s `\font` command still has a somewhat cumbersome syntax. Therefore, to allow the use of commands more in line with $\text{L}^{\text{A}}\text{TeX}$'s NFSS syntax Will Robertson has developed his fontspec package. It offers a simple way to select font families in $\text{L}^{\text{A}}\text{TeX}$ for arbitrary fonts. In particular it lets you fully control the selection of advanced font features that are available in OpenType or TrueType fonts.

2.4.1 Usage

For basic use, no package options are required:

```
\usepackage{fontspec}% font selecting commands
\usepackage{xunicode}% unicode character macros
\usepackage{xltextra} % a few fixes and extras
```

Ross Moore's xunicode package is highly recommended, as it provides access $\text{L}^{\text{A}}\text{TeX}$'s various methods for accessing extra characters and accents (for example, `\%`, `\$`, `\textbullet`, `\"u`, and so on), plus many more unicode characters.

Will Robertson's xltextra package, which loads the fontspecxunicode packages, adds a couple of general improvements to $\text{L}^{\text{A}}\text{TeX}$ under $X_{\text{E}}\text{TeX}$. It also provides the `\XeTeX` macro to typeset the $\text{X}_{\text{E}}\text{TeX}$ logo by loading the metalogo package.

It is important to note that the babel package is not really supported. Many languages, such as Vietnamese, Greek, and Hebrew, might not work correctly. You might have more chance with Cyrillic and Latin-based languages, however—fontspec ensures at least that fonts should load correctly, but hyphenation and other matters are not guaranteed.

fontspec has a list of options:

cm-default The Latin Modern fonts are not loaded;
no-math The maths fonts are not changed;
no-config the configuration file `fontspec.cfg` is not loaded;
quiet fontspec's warnings will only be written in the log file and not on the console.

2.4.2 Latin Modern defaults

fontspec defines a new $\text{L}^{\text{A}}\text{TeX}$ font encoding to allow the Latin Modern fonts (which are Unicode-encoded) to be used by default. Indeed, it does not really make sense to have the legacy Computer Modern fonts in the Unicode-enabled $X_{\text{E}}\text{TeX}$. Note that fontspec also requires the euenc package to be installed.

The package option (`[cm-default]`) instructs fontinst to ignore the Latin Modern fonts and use TeX 's standard Computer Modern fonts instead. This might be useful on a system where the Latin Modern fonts are not installed.

2.4.3 Maths 'fiddling'

By default, fontspec adjusts $\text{L}^{\text{A}}\text{TeX}$'s default maths setup in order to maintain the correct Computer Modern symbols when the roman font changes. However, it will attempt to avoid doing this if another maths font package is loaded (such as mathpazo or Will's upcoming unicode-math package).

If you find that it is not correctly changing the maths font you should specify the `[no-math]` package option to suppress its maths font component.

You can customise any part of the fontspec interface, e.g., selecting features or scripts, by creating a file `fontspec.cfg`, which is automatically loaded by X_YTeX if it is found. The package option `[no-config]` suppresses loading this file.

Since the fontspec package is quite verbose with its warning messages, an “experienced” user, who knows what she is doing, can specify the `[quiet]` package option, which directs all warning messages to the transcript (`.log`) file *only*.

2.4.4 A first overview

fontspec is a quite complex package since it has to handle a lot of font features. A basic preamble set-up is shown below, to simply select some default document fonts. See the file `fontspec-example.tex` for a more detailed example.

```
\usepackage{fontspec}
\defaultfontfeatures{Scale=MatchLowercase}
\setmainfont[Mapping=tex-text]{Minion Pro}
\setsansfont[Mapping=tex-text]{Myriad Pro}
\setmonofont{Courier Std}
```

2.4.5 Font selection

```
\fontspec[FontFeatures]{Fontname}
```

This is the basic command of the fontspec package. It lets you select *Fontname* from a T_EX family. The optional argument *FontFeatures* is a comma-separated list of features (see Section 1.4.2 on page 11).

As our first example, look how easy it is to select the *Minion Pro* typeface with the fontspec package:

My first fontspec example.
My first fontspec example.
MY FIRST FONTSPEC EXAMPLE.
MY FIRST FONTSPEC EXAMPLE.
My first fontspec example.
My first fontspec example.
MY FIRST FONTSPEC EXAMPLE.
MY FIRST FONTSPEC EXAMPLE.

```
\usepackage{fontspec,xltxtra}
\providecommand\MyText
{My first fontspec example.\\}
\fontspec{Minion Pro} \MyText
{\itshape \MyText}
{\scshape \MyText}
{\scshape\itshape \MyText}
\bfseries \MyText
{\itshape \MyText}
{\scshape \MyText}
{\itshape\scshape \MyText}
```

Exa.
2-4-1

The fontspec package takes care *automatically* of the necessary font definitions for those shapes as shown above. Furthermore, it is not necessary to install the font for X_YTeX specifically: every font that is installed in the operating system may be accessed.

2.4.6 Default font families

The `\setmainfont`, `\setsansfont`, and `\setmonofont` commands are used to select the default font families for the entire document. They take the same arguments as `\fontspec`, for instance:

Exa.
2-4-2

<p>Famous quick and jumping brown foxes. Famous quick and jumping brown foxes. Famous quick and jumping brown foxes.</p>	<pre>\usepackage{fontspec,xltxtra} \providecommand\MyText {Famous quick and jumping brown foxes.} \setmainfont{Adobe Garamond Pro} \setsansfont[Scale=0.86]{Cronos Pro} \setmonofont[Scale=0.8]{News Gothic Std} \rmfamily\MyText\par \sffamily\MyText\par \ttfamily\MyText</pre>
--	---

Here, the scales of the fonts have been chosen to equalise their lowercase letter heights. The `Scale` font feature also allows for automatic scaling, as will be explained later.

A more complex example which shows the italic and bold variants follows.

Exa.
2-4-3

<p>Famous quick and jumping brown foxes. <i>Famous quick and jumping brown foxes.</i> Famous quick and jumping brown foxes. <i>Famous quick and jumping brown foxes.</i> Famous quick and jumping brown foxes. <i>Famous quick and jumping brown foxes.</i> Famous quick and jumping brown foxes. <i>Famous quick and jumping brown foxes.</i> Famous quick and jumping brown foxes. <i>Famous quick and jumping brown foxes.</i> Famous quick and jumping brown foxes. <i>Famous quick and jumping brown foxes.</i></p>	<pre>\usepackage{fontspec,xltxtra} \providecommand\MyText {Famous quick and jumping brown foxes.} \setmainfont{Adobe Garamond Pro} \setsansfont[Scale=0.86]{Cronos Pro} \setmonofont[Scale=0.8]{News Gothic Std} \rmfamily\MyText\par {\itshape\MyText}\par {\bfseries\MyText}\par {\itshape\bfseries\MyText}\par \sffamily\MyText\par {\itshape\MyText}\par {\bfseries\MyText}\par {\itshape\bfseries\MyText}\par \ttfamily\MyText\par {\itshape\MyText}\par {\bfseries\MyText}\par {\itshape\bfseries\MyText}\par</pre>
---	---

Since `fontspec` has to parse and process its arguments at each call it can be more efficient to create a font instance for a given set of features using the `\newfontfamily` command, which creates commands that can be used like `\rmfamily`, `\sffamily`, etc.

Exa.
2-4-4

<p>The perfect match is hard to find. L O G O F O N T</p>	<pre>\usepackage{fontspec} \setmainfont{Georgia} \newfontfamily\lc[Scale=MatchLowercase]{Verdana} The perfect match {\lc is hard to find.}\ \newfontfamily\uc[Scale=MatchUppercase]{Arial} L O G O \uc F O N T</pre>
---	--

For cases where only one specific font face is needed, without accompanying italic or bold variants, the `\newfontface` command is available. In particular, this command can be useful when a font is of a fancy nature, e.g., it contains script or swash features that are only available in an italic variant, and

not in upright, etc.

*Characters [*349@!?] of a Brushy Nature.*

```
\usepackage{fontspec}
\newfontface\Brush{Brush Script Std Medium}
\Brush Characters [*349@!?] of a Brushy Nature.
```

Exa.
2-4-5

Automatic selection of bold, italic, and bold italic for certain fonts might not be adequate, in particular if the given font does not exist in bold or italic variants. Nevertheless, in such cases the user might want to choose matching shapes from a completely different font. In other instances a font can have a range of bold and italic fonts to choose between. The `BoldFont` and `ItalicFont` features are provided for these situations. If only one of these is used, the bold italic font is requested as the default from the *new font*.

Helvetica Neue Ultra Light

Helvetica Neue Ultra Light (italic)

Helvetica Neue Roman (bold)

Helvetica Neue Roman (bold italic)

```
\usepackage{fontspec}
\fontspec[BoldFont={Helvetica Neue 55 Roman}]
{Helvetica Neue 25 Ultra Light}
{Helvetica Neue Ultra Light} \
{\itshape Helvetica Neue Ultra Light (italic)} \
{\bfseries Helvetica Neue Roman (bold)} \
{\bfseries\itshape Helvetica Neue Roman (bold italic)} \
```

Exa.
2-4-6

In this example we want to use the font *Helvetica Neue 25 Ultra Light* (its full name has to be specified to the ICU processor), which has no bold variant, hence we tell `fontspec` to use *Helvetica Neue 55 Roman* when constructing the bold variants. We can also specify an explicit bold italic variant with the `BoldItalicFont` feature.

Fontspec: Chinese, Mandarin
(Simplified):

人人生而自由,在尊严和权利
上一律平等。

X_YTeX: Chinese, Mandarin
(Traditional):

人人生而自由,在尊嚴和權
利上一律平等。

And now the same vertically

利人	利人
上上	上上
一一	一一
律律	律律
平平	平平
等等	等等
。	。
，	，
在	在
尊	尊
严	严
和	和
权	权

```
\usepackage{fontspec,xltxtra,graphicx}
```

```
\XeTeXlinebreaklocale "zh" % allow linebreaks
\XeTeXlinebreakskip = 0pt plus 1pt minus 0.1pt
\setmainfont[Mapping=tex-text]{Minion Pro}
\providecommand{\ZHS}{%
人人生而自由,在尊严和权利上一律平等。}
\providecommand{\ZHT}{%
人人生而自由,在尊嚴和權利上一律平等。}
%%%% Use font MingLiU with 'vert' feature
\parbox{45mm}{\raggedright
Fontspec: Chinese, Mandarin (Simplified):\
\fontspec{MingLiU} \ZHS \
\rmfamily
\XeTeX: Chinese, Mandarin (Traditional):\
%%%% Define font in plain xetex
\font\body="MingLiU" \body \ZHT } \[3mm]
%%%% Rotate glyphs
\rmfamily And now the same vertically\
\fontspec[Vertical=RotatedGlyphs]{MingLiU}
\quad\rotatebox{-90}{\parbox{45mm}{\ZHS}}
\font\body="MingLiU:vertical" \body
\quad\rotatebox{-90}{\parbox{45mm}{\ZHT}}
```

Exa.
2-4-7

2.5 XeTeX and other engines

The two key features XeTeX offers are (a) native support for Unicode, including complex non-Latin scripts, and (b) easy use of modern font formats (TrueType and OpenType).

Earlier, Unicode support was offered by Omega (and then Aleph); more recently, this has been incorporated into LuaTeX, which also has support for direct use of OpenType fonts. Nevertheless, according to Jonathan Kew¹ there are major differences in the approach taken by the different projects, in particular,

XeTeX values	LuaTeX (and predecessors)
ease of setup and use	ultimate flexibility
uses available libraries	control every aspect of the implementation
wherever feasible do “the right thing” automatically	provide authors or macro writers with low-level tools

¹Presentation at BachoTeX2008 (<http://www.gust.org.pl/bachotex/2008/presentations/XeTeX-BachoTeX2008-pres.pdf>).

Handling all those scripts

3.1 Writing systems	49
3.2 Bidirectional typesetting	55
3.3 Languages using the Arabic alphabet.	61
3.4 Typesetting Chinese	79
3.5 Examples of the use of Unicode	88

As shown in Figures 2.2 and 2.3 on page 20, the world has many scripts. In this chapter we first present a brief overview of the world’s writing systems. Problems related to bidirectional typesetting and their solution are described in Section 3.2. Application packages for Arabic and Chinese typesetting are the subject of Sections 3.3.2 and 3.4, respectively. Finally, in Section 3.5 we give hints about where to find information on Unicode fonts and freely available texts in UTF-8.

3.1 Writing systems

It is accepted that every human community possesses language, yet the development and adoption of writing systems occurred only quite recently in the history of mankind. Moreover, writing systems, once they are introduced, generally change rather more slowly than the spoken variant they represent, and they thus often preserve features and expressions which are no longer current in the spoken language. Nevertheless, the great benefit of writing systems is that they maintain a persistent record of information expressed in a language, which can be retrieved independently of the initial act of formulation.

Writing systems require:

- a set of defined base elements or symbols, individually termed characters or graphemes, and collectively called a script;
- a set of rules and conventions understood and shared by a community, which arbitrarily assign meaning to the base elements, their ordering, and relations to one another;
- a language (generally a spoken language) whose constructions are represented and able to be recalled by the interpretation of these elements and rules;
- some physical means of distinctly representing the symbols by application to a permanent or semi-permanent medium, so that they may be interpreted (usually visually, but tactile systems have also

been devised).

3.1.1 Basic terminology

The study of writing systems has developed along partially independent lines in the examination of individual scripts, and as such the terminology employed differs somewhat from field to field [6].

The generic term *text* may be used to refer to an individual product of a writing system. The act of composing a text may be referred to as *writing*, and the act of interpreting the text as *reading*. In the study of writing systems, *orthography* refers to the method and rules of observed writing structure (literal meaning, “correct writing”), and in particular for *alphabetic* systems, includes the concept of *spelling*.

Graphemes are the atomic units of a given writing system, i.e., the minimally significant elements which taken together comprise the set of “building blocks” out of which texts of a given writing system may be constructed, along with rules of correspondence and use. For example, for standard contemporary English graphemes include the uppercase and lowercase forms of the twenty-six letters of the Latin alphabet (corresponding to various *phonemes* the atoms of the spoken language), marks of punctuation (mostly non-phonemic), and a few other symbols such as those for numerals (logograms for numbers).

A given grapheme may be represented in a wide variety of ways, each variation being visually distinct in some regard, but all are interpreted as representing the “same” grapheme. These individual variations are known as allographs of a grapheme, e.g., the lowercase letter “a” has different allographs depending on the medium used, the writing instrument, the stylistic choice of the writer, and an individual’s handwriting.

The terms *glyph*, *sign*, and *character* are sometimes used to refer to a grapheme. The glyphs of most writing systems are made up of lines (or strokes) and are therefore called linear, but there are glyphs in non-linear writing systems made up of other types of marks, such as Cuneiform and Braille.

Writing systems are conceptual systems, as are the languages to which they refer. Writing systems may be regarded as complete according to the extent to which they are able to represent all that may be expressed in the spoken language.

3.1.2 History of writing systems

http://en.wikipedia.org/wiki/History_of_writing

Writing systems were preceded by proto-writing, systems of *ideographic* (representing an idea) or early mnemonic (serving as a memory aid) symbols, e.g., the Jiahu Script (ca 6600 BCE, tortoise shells, China), the Vinca script (ca. 4500 BCE, Tărtăria tablets, Romania), and the early Indus Harappan script (ca. 3500 BC, N-W India).

The invention of the first writing systems is roughly contemporary with the beginning of the Early Bronze Age in the late Neolithic (around 3000 BCE), e.g., the Sumerian archaic cuneiform script and the Egyptian hieroglyphs, generally considered the earliest writing systems, both emerge out of their ancestral proto-literate symbol systems as the first coherent texts from about 2600 BCE. Similarly, the Chinese script is considered to have developed independently of the Middle Eastern scripts mentioned previously, around 1600 BCE.

It is generally accepted that the first true alphabetic writing appeared in the Middle Bronze Age (2000–1500 BCE), as a representation of language developed by Semitic workers in Central Egypt.¹ Over the next five centuries it spread north, and all subsequent alphabets around the world have either descended from it, many via the Phoenician alphabet, or were directly inspired by its design.

The first purely alphabetic script is thought to have been developed around 2000 BCE for Semitic workers in central Egypt.

¹“History of the alphabet”, see http://en.wikipedia.org/wiki/History_of_the_alphabet.

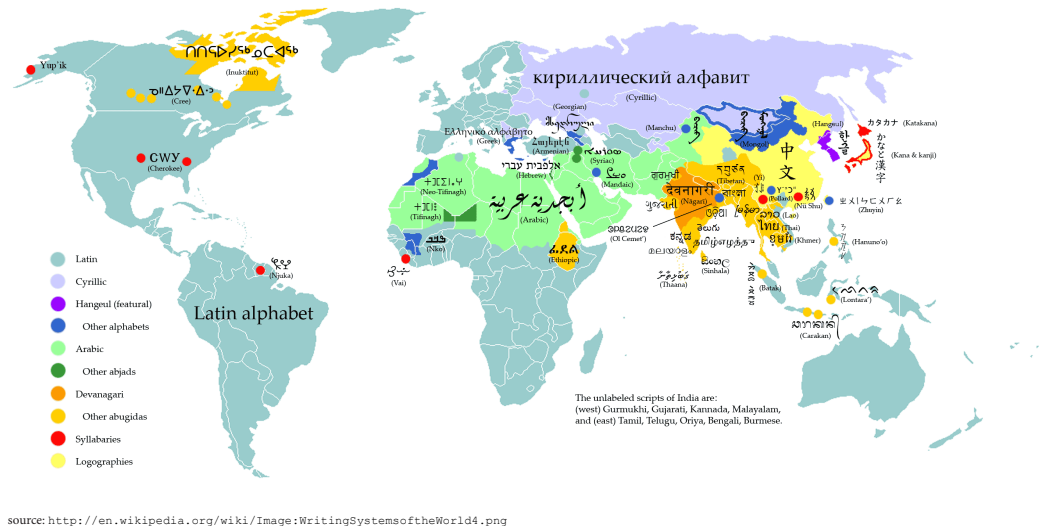


Figure 3.1: Writing systems used in the world today

3.1.3 Types of writing systems

Figure 3.1 shows the writing systems and their types as they are used in the world today.

The oldest-known forms of writing were mainly of the *logographic* type, i.e., they used a single grapheme for representing a morpheme, the atomic unit of meaning in a language.¹ Such forms combined *pictographic* (a symbol representing a concept, object, activity, place, event, etc. by a *drawing*) and *ideographic* (a symbol representing an *idea*) elements.

Most writing systems can be broadly divided into three categories, namely *logographic*, *syllabic*, and *alphabetic*, although a given writing system can contain two, or all three, in which case one often talks of a *complex system*.

Various types of writing systems exist.

- a *logographic* symbol represents a *morpheme* (e.g., Chinese characters);
- a *syllabic* type symbol represents a *syllable* (e.g., Japanese *kana*);
- an *alphabetic* type symbol represents a *phoneme*: consonant or vowel (e.g., Latin alphabet);
- an *abugida* type symbol represents a *phoneme*: consonant+vowel (e.g., Indian Devanagari);
- an *abjad* type symbol represents a *phoneme*: consonant (e.g., Arabic alphabet);
- a *featural* type symbol represents a *phonetic feature* (e.g., Korean hangul).

3.1.3.1 Logographic writing systems

A *logogram* (see <http://en.wikipedia.org/wiki/Logogram>) is a single written character which represents a complete grammatical word or *morpheme*. Thus, many logograms are required to write all the words of language. The vast array of logograms and the memorization of what they mean are the major disadvantage of the logographic systems over alphabetic systems. On the other hand, since the meaning is inherent to the symbol, the same logographic system can theoretically be used to represent different languages. In practice, this is only true for closely related languages, like the various dialects

¹See <http://en.wikipedia.org/wiki/Morpheme>.

of the Chinese language. Speakers of dialects of the various provinces of China will understand the characters of given Chinese text but pronounce them in quite different, and sometimes mutually unintelligible, ways. Furthermore, Japanese uses Chinese logograms extensively in its writing systems, with most of the symbols carrying the same or similar meanings. However, the semantics, and especially the grammar, are different enough that a long Chinese text is not readily understandable to a Japanese reader without any knowledge of basic Chinese grammar, though short and concise phrases such as those on signs and newspaper headlines are much easier to comprehend.

While most languages do not use wholly logographic writing systems many languages use some logograms. A good example of modern western logograms are the Hindu-Arabic numerals — everyone who uses those symbols understands what “1” means, whether the symbol is pronounced as “one”, “un”, “eins”, “yi”, “odin”, “ichi”, or “ehad”. Other western logograms include the ampersand “&” (used for *and*), the “@” (with its many semantic uses), the “%” (as percent), and many currency symbols (\$, ¢, £, ¥, €, etc.).

Logograms are sometimes called *ideograms*, symbols which graphically represent abstract ideas, but this use is somewhat inappropriate for Chinese characters since they often consist of semantic–phonetic compounds, i.e., they include an element that represents the meaning and another that represents the pronunciation.

Today the only surviving important modern logographic writing system is the Chinese one, whose characters are or were used, with varying degrees of modification, in Chinese, Japanese, Korean, Vietnamese, and other east Asian languages. Ancient Egyptian hieroglyphics and the Mayan writing system are also systems with certain logographic features, although they have marked phonetic features as well, and they are no longer in current use.

3.1.3.2 Syllabic writing systems

A syllabary (see also <http://en.wikipedia.org/wiki/Syllabary>) is a set of written symbols that represent (or approximate) syllables, which make up words. A symbol in a syllabary typically represents a consonant sound followed by a vowel sound, or just a vowel alone. In a true syllabary there is no systematic graphic similarity between phonetically related characters.¹

Syllabaries are best suited to languages with relatively simple syllable structure, such as Japanese, where the number of possible syllables is no more than about fifty to sixty. In contrast, English would need many thousands to represent all its possible syllable structures. The Japanese language uses Chinese *Kanji*, as well as two syllabaries together called *kana*, namely *hiragana* and *katakana* (developed around 700 CE). They are mainly used to write some native words and grammatical elements, as well as foreign words (see *Japanese writing system* http://en.wikipedia.org/wiki/Japanese_writing_system)

Languages that use syllabic writing include Mycenaean Greek (Linear B), the Native American language Cherokee, the African language Vai, the English-based creole language Ndyuka (the Afaka script), Yi language in China, the Nü Shu syllabary for Yao people, China, and the ancient Filipino script Alibata. The Chinese, Cuneiform, and Maya scripts are largely syllabic in nature, although based on logograms. They are therefore sometimes referred to as logosyllabic.

3.1.3.3 Alphabetic writing systems

An alphabet (see <http://en.wikipedia.org/wiki/Alphabet>) is a small set of *letters* — basic written symbols — each of which roughly represents a *phoneme* of a spoken language (as it is currently pronounced or as it was pronounced in the past).

¹Some syllabaries exhibit a graphic similarity for the vowels. For instance in hiragana, the characters for “ke”, “ka”, and “ko” show no graphical similarity to indicate their common “k” phonetic element. This is in contrast to *abugida*, where each grapheme typically represents a syllable but where characters representing related sounds are similar graphically, i.e., a common consonantal base is annotated in a more or less consistent manner to represent the vowel in the syllable

In a perfectly phonemic alphabet, the phonemes and letters would correspond perfectly in two directions: a writer could predict the spelling of a word given its pronunciation, and a speaker could predict the pronunciation of a word given its spelling. Examples of languages with such an alphabet are Serbocroatian or Finnish, and these have much lower barriers to literacy than languages such as English, which has a very complex and irregular spelling system, which has hardly evolved since many centuries, whereas the spoken language has considerably. Moreover, since writing systems have been borrowed for languages they were not designed for, the degree to which letters of an alphabet correspond to phonemes of a language varies greatly from one language to another and even within a single language. Although possible, using a truly phonetic alphabet (e.g., the International Phonetic Alphabet (IPA), see http://en.wikipedia.org/wiki/International_Phonetic_Alphabet) for a natural spoken language would be very cumbersome, as it would have to have a huge variety of phonetic variation.

3.1.3.4 Abjads

The first type of alphabet that was developed was the abjad, an alphabetic writing system which uses one symbol per consonant, vowels usually not being marked (see <http://en.wikipedia.org/wiki/Abjad>).

Almost all abjad scripts are used for Semitic languages and the related Berber languages which have a morphemic structure which makes the denotation of vowels redundant in most cases.

Some abjads (e.g., Arabic and Hebrew) have markings for vowels as well (in this case they are called “impure” abjads), although they most only use them in special contexts, such as for teaching. On the other hand, when an abjad script was adapted to a non-Semitic language the derived abjad has been extended with vowel symbols to become full alphabets, the most famous case being the derivation of the Greek alphabet from the Phoenician abjad.

3.1.3.5 Abugida

An abugida (see <http://en.wikipedia.org/wiki/Abugida>) is an alphabetic writing system in which each letter (basic character) represents a consonant accompanied by a specific vowel; other vowels are indicated by modification of the consonant sign, either by means of diacritics or through a change in the form of the consonant. In some abugidas, the absence of a vowel is indicated overtly. About half the writing systems in the world, including the various scripts used for most Indo-Aryan languages, are abugidas.

For instance, in an abugida there is no sign for “k”, but instead one for “ka”, the “a” being *inherent* vowel. The phoneme “ke” is written by modifying the “ka” sign in a way that is consistent with how one would modify “la” to get “le”. In many abugidas the modification is the addition of a vowel sign, but other possibilities are imaginable (and used), such as rotation of the basic sign, addition of diacritical marks, and so on (an example can be seen for three Indic scripts in Figure 3.1. More information on Indic languages can be found at the Web page <http://www.unicode.org/notes/tn10/indic-overview.pdf>).

3.1.3.6 Featural writing systems

A featural script represents finer detail than an alphabet. Here symbols do not represent whole phonemes, but rather the elements (features) that make up the phonemes, such as *voicing* or its *place of articulation*. The only prominent example is Korean Hangul, where the featural symbols are combined into alphabetic letters, and these letters are in turn joined into syllabic blocks, so that the system combines three levels of phonological representation.

Table 3.1: Indic consonant–vowel combinations in various Indic abugidas

<i>position</i>	<i>syllable</i>	<i>pronunciation</i>	<i>derived from</i>	<i>script</i>
above	के	/ke:/	क /k(a)/	Devanagari
below	कु	/ku/		
left	कि	/ki/		
right	को	/ko:/		
around	கௌ	/kau/	க /ka/	Tamil
within	ಕಿ	/ki/	ಕೆ /ka/	Kannada

Exa.
3-1-1

3.1.4 Language Resources

<http://www.geonames.de/>

This website provides a treasure of data in many languages and scripts. It provides tables with the countries of the world in their own languages and scripts, with official names, capitals, flags, coats of arms, administrative divisions, national anthems, and translations of the countries and capitals. Also available are translations of the names of the days, months, planets, geographical names, such as rivers, mountains, etc., chemical elements, religions, numbers, and an extended glossary with several hundred words translated into languages classified per family.

<http://www.lexilogos.com/>

Information (in French) on many languages, with examples of phrases.

3.1.5 Freely available Unicode encoded fonts

The site “Wazu japan’s Gallery of Unicode Fonts” (<http://www.wazu.jp/>) was created by David McCreedy and Mimi Weiss. Currently the site is maintained by Wazu Japan. The site displays samples of available Unicode fonts ordered by writing system (roughly speaking Unicode ranges). Luc Devroye’s web site (<http://cg.scs.carleton.ca/~luc/fonts.html>) also has a long list of free and shareware fonts classified by language.

3.1.6 Directionality

Different scripts are written in different directions. The early alphabet could be written in any direction: either horizontal (left-to-right or right-to-left) or vertical (up or down). It could also be written *boustrophedon*: starting horizontally in one direction, then turning at the end of the line and reversing direction. Egyptian hieroglyph is one such script, where the beginning of a line written horizontally was to be indicated by the direction in which animal and human ideograms are looking.

The Greek alphabet and its successors settled on a left-to-right pattern, from the top to the bottom of the page. Other scripts, such as Arabic and Hebrew, came to be written right-to-left. Scripts that incorporate Chinese characters have traditionally been written vertically (top-to-bottom), from the right to the left of the page, but nowadays are frequently written left-to-right, top-to-bottom, due to Western influences, a growing need to accommodate terms in the Roman alphabet, and technical limitations in popular electronic document formats. The Mongolian alphabet is unique in being the only script written top-to-bottom, left-to-right; this direction originated from an ancestral Semitic di-

rection by rotating the page 90° counter-clockwise to conform to the appearance of Chinese writing. Scripts with lines written away from the writer, from bottom to top, also exist, such as several used in the Philippines and Indonesia.

3.1.7 Writing systems on computers

Different ISO/IEC standards are defined to deal with each individual writing systems to implement them in computers (or in electronic form). Today most of those standards are re-defined in a better collective standard, the ISO 10646, also known as Unicode. In Unicode, each character, in every language's writing system, is in principle given a unique identification number, known as its code point. The computer's software uses the code point to look up the appropriate character in the font file, so the characters can be displayed on the page or screen.

3.2 Bidirectional typesetting

Vafa Khalighi's (vafa@users.berlios.de) `bidi` package provides a convenient interface for typesetting bidirectional texts with $\text{X}_{\text{e}}\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ ¹.

This section is intended for people who use `bidi` directly, people who use other packages that depend on `bidi`, and developers of the packages that depend on `bidi`.

`bidi` modifies lots of $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ classes and packages so that you can use them for your bidirectional typesetting. `bidi` currently supports the standard $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ kernel, the `amsart`, `amsbook`, `article`, `bidibeamer` (modified version of the `beamer` class), `bidimemoir` (modified version of the `memoir` class), `bidimoderncv` (modified version of the `moderncv` class), `bidipresentation`, `book`, `bookest`, `extbook`, `rapport3`, `refrep`, `report`, `scrartcl`, `scrbook`, `scrreprt` classes, and the `amsthm`, `array`, `booktabs`, `beamerthemebidiJLTree` (modified version of the `beamerthemeJLTree` package), `bidi2in1`, `bidibeamerbaseauxtemplates` (modified version of `beamerbaseauxtemplates` package), `bidibeamerbasemplates` (modified version of the `beamerbasemplates` package), `cvthemebidicasual` (modified version of `cvthemecasual` package), `cvthemebidiclassic` (modified version of the `cvthemeclassic` package), `dcolumn`, `draftwatermark`, `fancyhdr`, `graphicx`, `hhline`, `listings`, `longtable`, `minitoc`, `multirow`, `pdfpages`, `pstricks`, `ragged2e`, `stabular`, `supertabular`, `tabls`, `tabularx`, `tabulary`, `threeparttable`, `tikz`, `tocloft`, `tocstyle` and `wrapfig` packages. Anything else is not supported yet but this does not mean they will not work with `bidi`, please feel free to experiment using other packages and classes with `bidi` but please note that you are on your own. In future versions of the `bidi` package, more classes and packages will be supported.

3.2.1 Using The `bidi` Package

You can use the package by simply putting `\usepackage{bidi}` in the preamble of your document. When using `bidi` the following should be noted.

1. The `bidi` package automatically loads the `amsmath` package so that you do not need to load it yourself.
2. The `bidi` package should be the last package that you load in the preamble of your document. This is because `bidi` modifies lots of commands defined in other $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ packages so that they can be used for bidirectional typesetting. If you do not load the `bidi` package as your last package, the `bidi` definitions would be overwritten and consequently you would not get the result you expect.

¹In fact, `bidi` can be used with any e- $\text{T}_{\text{E}}\text{X}$ -based engine, notably `PDF $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$` .

3. There is an exception to the above statement, you should always load package `xunicode` after¹ `bidi`. If you forget to follow this rule you will get an error message which looks like this:

```
! Package bidi Error: Oops! you have loaded package xunicode before
bidi package. Please load package xunicode after bidi package, and
then try to run xelatex on your document again.
```

See the `bidi` package documentation for explanation.

Type `H` <return> for immediate help.

...

```
1.4 \begin{document}
```

```
?
```

3.2.1.1 Package options

There are two options `RTLdocument` and `rldocument` which are essentially equivalent. They are intended mainly for RTL typesetting with some LTR typesetting and automatically activate `\setRTL`, `\RTLdblcol` and `\autofooterule` which are explained later.

3.2.2 Basic Direction Switching

`bidi` provides some commands, environments for direction switching:

3.2.2.1 Commands for direction switching

<pre>\setRTL \setRL \unsetLTR \setLTR \setLR \unsetRTL \unsetRL</pre>

The commands in the first row allows you to have RTL typesetting and the commands in the second row allows you to have LTR typesetting.

tesept si hcihw hpararap LTR a si sihT
 .tfel ot thgir morf

And this is an LTR paragraph which is typeset from left to right. Note the blank line that we put before changing the direction of typesetting.

```
\usepackage{bidi}
```

```
\setRTL
```

This is a RTL paragraph which is typeset from right to left.

```
\setLTR
```

And this is an LTR paragraph which is typeset from left to right. Note the blank line that we put before changing the direction of typesetting.

Exa.
3-2-1

¹This is because `amsmath` should be loaded before the `xunicode` package and `bidi` already loads `amsmath`. Hopefully this will change in future versions of the `bidi` package.

3.2.2.2 Environments for direction switching

```
\begin{RTL} ... \end{RTL}
\begin{LTR} ... \end{LTR}
```

The first environment allows you to have RTL typesetting and the second environment allows you to have LTR typesetting.

Exa.
3-2-2

<pre>tesepty si hcihw hpargarap LTR na si sihT .tfel ot thgir morf This is an LTR paragraph inside an RTL para- graph. ecno edom LTR ni gnittesepty era ew ereH .erom</pre>	<pre>\usepackage{bidi} \begin{RTL} This is an RTL paragraph which is typeset from right to left. \begin{LTR} This is an LTR paragraph inside an RTL paragraph. \end{LTR} Here we are typesetting in RTL mode once more. \end{RTL}</pre>
---	---

3.2.3 Typesetting Short RTL and LTR texts

```
\RLE{...} \RL{...}
\LRE{...} \LR{...}
```

The commands in the first row allow you to typeset a short piece of text from right to left and the commands in the second row allow you to typeset a short piece of text from left to right.

```
\usepackage{bidi}
\setRTL
This is an RTL paragraph and \LRE{these words} appeared LTR.

\setLTR
This is an LTR paragraph and \RL{these words sentence} appeared RTL.
```

Exa.
3-2-3

```
.RTL deraeppa these words dna hpargarap LTR na si sihT

This is an LTR paragraph and ecnetnes sdrow eseht appeared RTL.
```

3.2.4 Multicolumn Typesetting

3.2.4.1 Two column typesetting

```
\RTLdblcol \LTRdblcol
```

`\RTLdblcol` allows you to have RTL two column typesetting and `\LTRdblcol` allows you to have LTR two column typesetting as the options of the class file.

3.2.4.2 Multicolumn typesetting

For RTL multicolumn typesetting, you can use `fmultico` package which has the same syntax as `multicol` package.

```
\usepackage{bidi, fmultico}
\setRTL
\begin{multicols}{3}
```

```
EETS was founded in 1864 by Frederick James Furnivall, with the help
of Richard Morris, Walter Skeat, and others, to bring the mass of
unprinted Early English literature within the reach of students. It
was also intended to provide accurate texts from which the New (later
Oxford) English Dictionary could quote; the ongoing work on the
revision of that Dictionary is still heavily dependent on the
Society's editions, as are the Middle English Dictionary and the
Toronto Dictionary of Old English.
\end{multicols}
```

-vaeH llits si yranoitciD taht	saw tI .stneduts fo hcaer eht	4681 ni dednuof saw STEE
-icoS eht no tnedneped yli	-ucca edivorp ot dednetni osla	,llavinruF semaJ kcirederF yb
-diM eht era sa ,snoitide s'yte	weN eht hcihw morf stxet etar	-roM drahciR fo pleh eht htiw
eht dna yranoitciD hsilgnE eld	-oitciD hsilgnE)drofxO retal(,srehto dna ,taekS retlaW ,sir
-nE dIO fo yranoitciD otnoroT	-ogno eht ;etouq dluoc yran	detnirpnu fo ssam eht gnirb ot
.hsilg	fo noisiver eht no krow gni	nihtiw erutaretil hsilgnE yIraE

Exa.
3-2-4

You also can use `vwcol` package for RTL multicolumn typesetting.

```
\usepackage{bidi,vwcol}
\setRTL
\begin{vwcol}[widths={0.3,0.2,0.5},rule=2pt]
EETS was founded in 1864 by Frederick James Furnivall, with the help
of Richard Morris, Walter Skeat, and others, to bring the mass of
unprinted Early English literature within the reach of students. It
was also intended to provide accurate texts from which the New (later
Oxford) English Dictionary could quote; the ongoing work on the
revision of that Dictionary is still heavily dependent on the
Society's editions, as are the Middle English Dictionary and the
Toronto Dictionary of Old English.
\end{vwcol}
```

hsilgnE)drofxO retal(weN eht hcihw morf	-til hsilgnE yIraE	ni dednuof saw STEE
no krow gniogno eht ;etouq dluoc yranoitciD	eht nihtiw erutare	semaJ kcirederF yb 4681
-vaeH llits si yranoitciD taht fo noisiver eht	.stneduts fo hcaer	pleh eht htiw ,llavinruF
sa ,snoitide s'yteicoS eht no tnedneped yli	-ni osla saw tI	retlaW ,sirroM drahciR fo
eht dna yranoitciD hsilgnE elddiM eht era	edivorp ot dednet	gnirb ot ,srehto dna ,taekS
.hsilgnE dIO fo yranoitciD otnoroT	stxet etarucca	detnirpnu fo ssam eht

Exa.
3-2-5

3.2.5 More peculiarities for RTL typesetting

3.2.5.1 Handling color

Due to $\text{X}_{\text{L}}\text{T}_{\text{E}}\text{X}$'s limitations in handling colors, you cannot use the `color` and `xcolor` packages for generating RTL color texts. Instead you should use the `xecolour` package.

3.2.5.2 RTL cases

`\rcases` is defined in `bidi` for typesetting RTL cases.

Exa.
3-2-6

$\left. \begin{array}{l} \text{nem} \\ \text{nemow} \end{array} \right\} \text{sgnieB snamuH}$	<pre> \usepackage{bidi} \setRTL \[\rcases{\text{men}\cr\text{women}} \text{Humans Beings} \]</pre>
--	--

3.2.5.3 Footnotes

```

\footnote{...} \LTRfootnote{...} \RTLfootnote{...}
\setfootnoteRL \setfootnoteLR \unsetfootnoteRL
```

- `\footnote` in RTL mode produces an RTL footnote while in LTR mode it produces an LTR footnote.
- `\LTRfootnote` will always produce an LTR footnote, independent on the current mode.
- `\RTLfootnote` will always produce an RTL footnote, independent on the current mode.
- Specifying a `\setfootnoteRL` command anywhere will make `\footnote` produce an RTL footnote.
- Specifying either a `\setfootnoteLR` or an `\unsetfootnoteRL` command anywhere will make `\footnote` produce an LTR footnote.

The behavior of footnote rules can also be controlled.

```

\autofootnoterule \rightfootnoterule
\leftfootnoterule \textwidthfootnoterule
```

- `\rightfootnoterule` will put footnote rule on the right-hand side.
- `\leftfootnoterule` will put footnote rule on the left-hand side.
- `\textwidthfootnoterule` will draw the footnote rule with a width equal to `\textwidth`.
- `\autofootnoterule` will draw the footnote rule right or left aligned based on the direction of the first footnote following the rule (i.e., put in the current page).

3.2.6 Tabular material in RTL mode

You can typeset any tabular material in RTL mode, as seen below.

C11–C12		C13–C14		C15–C16	
C21	C22	C23	C24	C25	C26
C31	C32	C33	C34	C35	C36
C41–C44				C45–C46	

61C–51C		41C–31C		21C–11C	
62C	52C	42C	32C	22C	12C
63C	53C	43C	33C	23C	13C
64C–54C		44C–14C			

```

\usepackage{bidi}
\providecommand\Mytable{%
\begin{tabular}{|l|c|r|r|c|l|}\hline
\multicolumn{2}{|l|}{C11--C12}
& \multicolumn{2}{c|}{C13--C14}
& \multicolumn{2}{r|}{C15--C16}\\ \hline
C21 & C22 & C23 & C24 & C25 & C26\\ \hline
\cline{2-2}\cline{4-4}\cline{6-6}
C31 & C32 & C33 & C34 & C35 & C36\\ \hline
\cline{1-1}\cline{3-3}\cline{5-5}
\multicolumn{4}{|l|}{C41--C44} & & \\
& \multicolumn{2}{|r|}{C45--C46}\\ \hline
\hline\hline
\end{tabular}}
\Mytable\l[lex]
\setRTL
\Mytable

```

Exa.
3-2-7

By comparing the top (typeset in LTR mode) and the bottom (typeset in RTL mode) tables it is seen that in RTL mode the columns are indeed typeset from right to left, e.g., the leftmost column becoming the rightmost, etc. This behavior includes the numbering of the columns, as used in the `\cline` command, where in RTL mode, e.g., `\cline{2-2}` refers to the second rightmost column. Note that the alignment indicators (l and r) in the `\begin{tabular}` and `\multicolumn` arguments play their usual role of aligning the material left and right adjusted, respectively. A more complex example is the following.

```

\usepackage{bidi}
\newcommand{\rb}[1]{\raisebox{1.5ex}[0mm]{#1}}
\setRTL
\begin{tabular}{|r||c|r|c|r|c|r|}
\hline
& \multicolumn{2}{c|}{6.15--7.15 pm} & \multicolumn{2}{c|}{7.20--8.20 pm}
& \multicolumn{2}{c|}{8.30--9.30 pm} \\ \cline{2-7}
& \rb{Day} & \rb{Subj.} & \rb{Subj.} & \rb{Subj.} & \rb{Subj.} & \rb{Room} \\ \hline\hline
& \rb{Mon.} & \rb{UNIX} & \rb{Comp. Ctr} & \rb{Fortran} & \rb{Hall A}
& \rb{Math.} & \rb{Hall A} \\ \hline
& \rb{Tues.} & \rb{LaTeX} & \rb{Conf.~Room} & \rb{Fortran} & \rb{Conf~Room}
& \rb{Math.} & \rb{Hall A} \\ \hline
& \rb{Wed.} & \rb{UNIX} & \rb{Comp. Ctr} & \rb{C} & \rb{Hall A}
& \rb{ComSci.} & \rb{Hall A} \\ \hline
& \multicolumn{2}{c|}{\rb{Miss Baker} & \rb{Ms. Clark} & \multicolumn{2}{c|}} \\ \cline{3-3}\cline{5-5}

```



```
\rb{Fri.} & \rb{\LaTeX} & Conf.~Room & \rb{C++} & Conf.~Room
& \multicolumn{2}{c|}{\rb{canceled}}\ \hline
\end{tabular}
```

Exa.
3-2-8

mp 03.9–03.8		mp 02.8–02.7		mp 51.7–51.6		yaD
rehcaeT	.jbuS	rehcaeT	.jbuS	rehcaeT	.jbuS	
mooR		mooR		mooR		
slliM .rM	.htaM	kralC .sM	nartroF	htimS .rD	XINU	.noM
A llaH		A llaH		rtC .pmoC		
lliM .rM	.htaM	kralC .sM	nartroF	rekaB ssiM	L ^A T _E X	.seuT
A llaH		mooR .fnoC		mooR .fnoC		
senoJ .rD	.icSmoC	senoJ .rD	C	htimS .rD	XINU	.deW
A llaH		A llaH		rtC .pmoC		
delecnac		kralC .sM	++C	rekaB ssiM	L ^A T _E X	.irF
		mooR .fnoC		mooR .fnoC		

You can get an idea of the many additional features that are available in the bidi by looking at the examples accompanying the bidi package.

3.3 Languages using the Arabic alphabet

The Arabic alphabet (see http://en.wikipedia.org/wiki/Arabic_alphabet) is after the Latin alphabet, the second-most widely used alphabet around the world. The alphabet was first used to write texts in Arabic, in particular the Qur’an, the holy book of Islam. With the spread of Islam, it came to be used to write many other languages, such as Persian, Urdu, Pashto, Baloch, Malay, Balti, Brahui, Panjabi (in Pakistan), Kashmiri, Sindhi (in Pakistan), Uyghur (in China), Kazakh (in China), Kyrgyz (in China), Azerbaijani (in Iran) and Kurdish in Iraq and Iran. In order to accommodate the needs of these (often non-semitic) languages, new letters and other symbols were added to the original alphabet.

Arabic is written from right to left, and is written in a cursive style of script. There are 28 basic letters in the Arabic alphabet. In analogy with the rich set of typefaces in the Roman alphabet, Arabic scripts [3] come in a number of different Arabic calligraphy styles (see Figure 3.2 for a few examples).

In the Arabic alphabet there are no distinct upper and lower case letter forms. Both printed and written Arabic are cursive, with most of the letters directly connected to the letter that immediately follows. There are some non-connecting letters that do not connect with the following letter, even in the middle of a word. Each individual letter can have up to four distinct forms, depending on the position of the letter within a word or group of letters, as follows:

- *Initial*: beginning of a word; or in the middle of a word, following a non-connecting letter.
- *Medial*: between two connecting letters (non-connecting letters lack a medial form).
- *Final*: at the end of a word following a connecting letter.
- *Isolated*: at the end of a word following a non-connecting letter; or used independently.

Some letters appear almost the same in all four forms, while others display more variety. In addition, some letter combinations are written as ligatures (special shapes), including lam-alif. In many cases, dots will be placed above or below the central part of a letter to distinguish it from other similar letters.

The Arabic alphabet is an “impure” abjad since short vowels are not written, but long ones are. Therefore the reader must know the language in order to restore the vowels. However, in editions of the

Different styles of the phrase “In the name of God” (top to bottom):

- *Ruq'ah* or *Riqa'* is characterized by clipped letters composed of short straight lines and simple curves, as well as its straight and even lines of text. It is clear and legible and is the easiest script for daily handwriting. It is used in the titles of books and magazines, and in commercial advertisements.
- *Naskh*, *Naskhi* or *Nesih* is the most commonly used style for printing Arabic, and usually the first to be taught to children.
- *Nasta'liq* or *Nastaleeq* is one of the main genres of Islamic calligraphy. It has short verticals with no serifs, and long horizontal strokes. It is only used for titles and heading in writing Arabic, but a somewhat less elaborate version serves as the preferred style for writing Persian, Pashto and Urdu (and formerly for Ottoman Turkish)
- *Thuluth* is characterized by curved and oblique lines, with one-third of each letter sloping. It is a large and elegant, cursive script, used in medieval times on mosque decorations, and to write the heading of surahs, Qur'anic chapters.
- *Muhaqqaq* or *Muhakkak*, a now rarely used calligraphic script in Arabic derived from Thuluth by widening the horizontal sections of the letters in the Thuluth script.
- *Kufiq* or *Kufic* is the oldest calligraphic form of the various Arabic scripts. It was already in use at the time of the emergence of Islam so that the first copies of the Qur'an were written in this script. Kufic (the example shows *Square Kufiq*) is characterized by straight lines and angles, often with elongated verticals and horizontals.



source: www.islamicarchitecture.org/art/images/calligraphy/

Figure 3.2: Examples of six Arabic calligraphic styles

Qur'an or in didactic works vocalization marks are used, including a sign for vowel omission (sukūn) and one for gemination/doubling/lengthening of consonants (šadda).

3.3.1 Arab \TeX : Arabic typography with \TeX

Since 1992, when Klaus Lagally publicly released Version 2 of his arabtex package,¹ \TeX users have been able to typeset Arabic (and Hebrew) texts in a user-friendly way, and for many years Arab \TeX has become a standard typesetting tool for many Arabists. However, Lagally's masterful, but extremely complex, difficult to understand, and monolithic set of \TeX macros makes it at present a somewhat out-of-date piece of software. Arab \TeX performs all typesetting tasks, from parsing the input encoding, doing the contextual analysis, assembling the various forms of a character, and placing them on the page from right to left, by \TeX macros. Moreover, Arab \TeX can only be used with its specially designed fonts.

Today, with the advent of Unicode-encoded OpenType fonts, many of the formatting issues are encoded in the OpenType fonts and taken care of by the operating system. Therefore, a Unicode-based solution taking full advantage of the many nice Arabic OpenType fonts, is highly desirable. The Arab $\X_{\Omega}\TeX$ system, described in Section 3.3.2, is one way of solving the problem, while Youssef Jabri's arabi package [2] (available on CTAN in the directory /language/arabic/arabi/) provides an-

¹The URL <ftp://ftp.informatik.uni-stuttgart.de/pub/arabtex/arabtex.htm> gives information about the most recent version of the software (3.11, dated 2 July 2006, at the time of writing).

Table 3.2: Arab \TeX 's input conventions for Arabic and PersianExa.
3-3-1

a	ا	<i>a</i>	' <i>alif</i>	b	ب	<i>b</i>	<i>bā'</i>	p	پ	<i>p</i>	<i>pā'</i>
t	ت	<i>t</i>	<i>tā'</i>	_t	ث	<i>t</i>	<i>tā'</i>	^g	ج	<i>ǧ</i>	<i>ǧīm</i>
.h	ح	<i>ḥ</i>	<i>ḥā'</i>	_h	خ	<i>ḥ</i>	<i>ḥā'</i>	d	د	<i>d</i>	<i>dāl</i>
_d	ذ	<i>ḏ</i>	<i>ḏāl</i>	r	ر	<i>r</i>	<i>rā'</i>	z	ز	<i>z</i>	<i>zāy</i>
s	س	<i>s</i>	<i>sīn</i>	^s	ش	<i>š</i>	<i>šīn</i>	.s	ص	<i>ṣ</i>	<i>ṣād</i>
.d	ض	<i>ḏ</i>	<i>ḏād</i>	.t	ط	<i>ṭ</i>	<i>ṭā'</i>	.z	ظ	<i>ẓ</i>	<i>ẓā'</i>
'	ع	'	' <i>ayn</i>	.g	غ	<i>ǧ</i>	<i>ǧayn</i>	f	ف	<i>f</i>	<i>fā'</i>
q	ق	<i>q</i>	<i>qāf</i>	v	ف	<i>v</i>	<i>vā'</i>	k	ك	<i>k</i>	<i>kāf</i>
g	گ	<i>g</i>	<i>gāf</i>	l	ل	<i>l</i>	<i>lām</i>	m	م	<i>m</i>	<i>mīm</i>
n	ن	<i>n</i>	<i>nūn</i>	h	ه	<i>h</i>	<i>hā'</i>	w	و	<i>w</i>	<i>wāw</i>
y	ي	<i>y</i>	<i>yā'</i>	_A	ى	<i>ā</i>	' <i>alif</i> <i>maqṣūra</i>	T	ة	<i>h</i>	<i>tā'</i> <i>marbuṭa</i>

other.

Table 3.2 shows Arab \TeX 's input conventions for the Arabic and Persian languages.

A small example of the use of Arab \TeX is the following Arabic anecdote about Juha and the 10 donkeys (We will use the text of Example 3-3-2 also in the examples of Arab \TeX). The text is shown fully vocalized (`\fullvocalize`) and is transliterated inline (`\transtrue`). The title is centered and typeset in bold (`\setnashbf`). The short Arabic text of the title is marked up inside the characters sequence `\< and >`, while the longer Arabic text of the body of the story is enclosed inside an `arabtext` environment. Compare the typeset output with the input text using the input conventions of Table 3.2. Note the different forms of the letters, which are all composed by Arab \TeX 's macros.

```
\usepackage{arabtex, atrans, nashbf}
\setarab\transtrue\fullvocalize
\setnashbf \centerline {\<^gu.hA wa-.hamIruhu al-`a^saraTu>}
\transtrue\setnash
\begin{arabtext}
i^starY ^gu.hA `a^saraTa .hamIriN.
fari.ha bihA wa-sAqahA 'amAmahu,
_tumma rakiba wA.hidaN minhA.
wa-fI al-.t.tarIqi `adda .hamIrahu wa-huwa rAkibuN,
fa-wa^gadahA tis`aTaN.
_tumma nazala wa-`addahA fa-ra'AhA `a^saraTuN fa-qAla:
```

```
'am^sI wa-'aksibu .himAraN,
'af.dalu min 'an 'arkaba wa-'a_hsara .himAraN.
\end{arabtext}
```

جُحَا وَحَمِيرَةُ الْعَشْرَةُ *ḡuḥā wa-ḥamīruhu 'l-ašaratu*

*ištarā ḡuḥā 'ašarata ḥamīrin. fariḥa biḥā wa-sāqahā 'amāmahu, ṭumma rakiba wāḥidan minhā. wa-
fī 't-ṭarīqi 'adda ḥamīrahu wa-huwa*

إِشْتَرَى جُحَا عَشْرَةَ حَمِيرٍ فَرِحَ بِهَا وَسَاقَهَا أَمَامَهُ، ثُمَّ رَكِبَ وَاجِدًا مِنْهَا. وَفِي الطَّرِيقِ عَدَّ حَمِيرَهُ وَهُوَ
rākibun, fa-waḡadahā tīsatan. ṭumma nazala wa-addahā fa-ra'āḥā 'ašaratun fa-qāla:

رَاكِبٌ، فَوَجَدَهَا تِسْعَةً. ثُمَّ نَزَلَ وَعَدَّهَا فَرَاَهَا عَشْرَةً فَقَالَ:

amšī wa-'aksibu ḥimāran, afḍalu min 'an 'arkaba wa-'ahsara ḥimāran.

أَمْشِي وَأَكْسِبُ حِمَارًا، أَفْضَلُ مِنْ أَنْ أُرْكَبَ وَأَحْسَرَ حِمَارًا.

Exa.
3-3-2

3.3.2 ArabX_YT_EX: Arabic typography with X_YT_EX

François Charette's `arabxetex` package is a X_YL^AT_EX adaptation of Klaus Lagally's `arabtex` (see Section 3.3.1). The main advantage of the package is that it allows you to use all OpenType encoded Arabic fonts that you have available on your system. In particular, the package requires that you declare the default Arabic font, `\arabicfont`, with `fontspec`'s `\newfontfamily` command.

The `arabxetex` package consists of a set of T_ECkit mappings (see Section 2.2.5) for converting internally from `arabtex`'s ASCII input convention to Unicode, and a L^AT_EX style file (`arabxetex.sty`) that provides a convenient user interface for typesetting in those languages. With respect to `arabtex`'s conventions, `arabxetex` introduces several additions, and a few minor modifications (see the next section). `arabxetex` relies on the package `bidi` (see Section 3.2).

The `arabtex` input encoding

Apart from ease and legibility, the `arabtex` input conventions offer several advantages for typesetting in the Arabic script. As the examples in this section will show, indeed, it is straightforward to mix Unicode and `arabtex` encodings on input, and to switch between romanized transliteration and the Arabic script on output. This comes in handy when one wants to input L^AT_EX constructs inside Arabic sources or handle complex multi-layer documents, such as critical editions, where footnotes and annotations abound, and where dealing with a plain ASCII encoding is a genuine advantage, all the more so since ArabT_EX's input conventions allow you a full control of the typographical details.

Support for languages using the Arabic script

Languages supported at present are the same as in `arabtex`, namely: Arabic, Maghribi Arabic, Farsi (Persian), Urdu, Sindhi, Kashmiri, Ottoman Turkish, Kurdish, Jawi (Malay) and Uighur. `arabxetex` adds support for several additional Unicode characters, so that some more languages are probably supported de-facto as well (such as Western Punjabi).

For Arabic RL (from-right-to-left) texts the `arabxetex` package defines the `arab` environment — and the equivalent `\textarab` command for short Arabic text insertions inside left-to-right input texts.¹ For other languages written in the Arabic alphabet similar environments and commands, are available, as follows.

- `\begin{farsi}[opt]...\end{farsi}` `\farsi[opt]{...}`

¹Similarly, for left-to-right “Latin” insertions inside Arabic text the `\textroman` command can be used.

- `\begin{kashmiri}[opt]...\end{kashmiri}` `\kashmiri[opt]{...}`
- `\begin{kurdish}[opt]...\end{kurdish}` `\kurdish[opt]{...}`
- `\begin{malay}[opt]...\end{malay}` `\malay[opt]{...}`
- `\begin{ottoman}[opt]...\end{ottoman}` `\ottoman[opt]{...}`
- `\begin{pashto}[opt]...\end{pashto}` `\pashto[opt]{...}`
- `\begin{sindhi}[opt]...\end{sindhi}` `\sindhi[opt]{...}`
- `\begin{urdu}[opt]...\end{urdu}` `\urdu[opt]{...}`
- `\begin{uighur}[opt]...\end{uighur}` `\uighur[opt]{...}`

For some entries in this list alternative names exist, namely `persian` for `farsi`, `turk` for `ottoman`, and `jawi` for `malay`.

The optional argument `opt` in all of these commands or environments can take one or more of the following values. The equivalent command in Arab_XTeX is given between square brackets when it exists.

- novoc** *non-vocalized* mode: no diacritics are added (the default global option) [`\novocalize`].
- fullvoc** *fully vocalized*: mode every short vowel written will generate the corresponding diacritical mark [`\fullvocalize`].
- voc** *vocalized* mode: as `fullvoc`, but *sūkun* and *waṣla* will not be generated [`\vocalize`].
- trans** transliteration mode [`\transtrue`].
- utf** input in plain UTF-8 encoding. When not in transliteration mode, this option is in principle not strictly needed since one can mix Arab_XTeX's ASCII input conventions and UTF-8 input.

Transliteration

At present Arab_XTeX offers `arabtex` transliteration mappings for Arabic, Persian, Urdu, Sindhi and Pashto. It is foreseen to implement alternative transliteration conventions for each language, as with `arabtex`, e.g., ZDMG, Encyclopedia Iranica, etc. (a list of such schemes is at the URL <http://transliteration.eki.ee/pdf/Arabic.pdf>)

As with `arabtex` (see Example 3-3-2), the transliteration is by default typeset in italics. This can be customized with the `\SetTranslitStyle` command. In the transliteration one can capitalize proper names by prefixing the word with the command `\UC`, e.g.,

Exa.
3-3-3

al-shaykh al-‘ālim Naṣīr al-Dīn al-Ṭūsī

```
\usepackage{arabxetex}
\newfontfamily\arabicfont[Script=Arabic]{Scheherazade}
\newfontfamily\gentium{Gentium}
\SetTranslitStyle{\gentium\itshape}
\begin{arab}[trans]
al-^say_h al-^Alim \UC na.sIr \UC al-dIn \UC al-.tUsI
\end{arab}
```

Since the transliteration is coded in Unicode we must ensure that all needed Latin extension characters are available in the font. Therefore we used the font *gentium* in this example. Note also that in the transliteration, the article *al-* is automatically skipped.

Emphasis

In Arabic emphasis is often indicated with a line over the text to be highlighted. In Arab_XTeX this is achieved with the `\aemph` command. The following example shows how this works, first without

vocalization and then with vocalization.

مثال: ٤٥ درجة
مثال: ٤٥ دَرَجَة

```
\usepackage{arabxetex}
\newfontfamily\arabicfont[Script=Arabic,Scale=2.0]
    {Scheherazade}
\begin{arab}[novoc]mi_tAl: \aemph{45} darajaT\end{arab}
\begin{arab}[voc] mi_tAl: \aemph{45} darajaT\end{arab}
```

Exa.
3-3-4

ArabT_EX's four representation variants

The following somewhat longer example uses the same text as Example 3-3-2, but shows the four presentation variants introduced previously one after the other. We use the *Traditional Arabic* font as default Arabic font (`\arabicfont` command) and *Gentium* as font for the non-Arabic text (with the `\setmainfont` command, which sets the “main” font for the document).

```
\usepackage[no-math]{fontspec}
\setmainfont{Gentium} \usepackage{arabxetex} \newfontfamily\arabicfont
[Script=Arabic,Scale=1.2]{Traditional Arabic}
% Story of Juha and the 10 donkeys
\begin{arab}% No short vowels shown
\begin{center}\bfseries\large ^gu.hA wa-.hamIruhu al-`a^saraTu\end{center}
i^starY ^gu.hA `a^saraTa .hamIriN.
fari.ha bihA wa-sAqahA 'amAmahu,_tumma rakiba wA.hidaN minhA.
wa-fI al-.t.tarIqi `adda .hamIrahu wa-huwa rAkibuN, fa-wa^gadahA tis`aTaN.
_tumma nazala wa-`addahA fa-ra'AhA `a^saraTuN fa-qAla: \\\
'am^sI wa-'aksibu .himAraN, 'af.dalu min 'an 'arkaba wa-'a_hsara .himAraN.
\end{arab}
\begin{arab}[fullvoc]% All short vowels shown
\begin{center}\bfseries\large ^gu.hA wa-.hamIruhu al-`a^saraTu\end{center}
i^starY ^gu.hA `a^saraTa .hamIriN.
fari.ha bihA wa-sAqahA 'amAmahu,_tumma rakiba wA.hidaN minhA.
wa-fI al-.t.tarIqi `adda .hamIrahu wa-huwa rAkibuN, fa-wa^gadahA tis`aTaN.
_tumma nazala wa-`addahA fa-ra'AhA `a^saraTuN fa-qAla: \\\
'am^sI wa-'aksibu .himAraN, 'af.dalu min 'an 'arkaba wa-'a_hsara .himAraN.
\end{arab}
\begin{arab}[voc] % All short vowels shown except for sukun and wasla
\begin{center}\bfseries\large ^gu.hA wa-.hamIruhu al-`a^saraTu\end{center}
i^starY ^gu.hA `a^saraTa .hamIriN.
fari.ha bihA wa-sAqahA 'amAmahu,_tumma rakiba wA.hidaN minhA.
wa-fI al-.t.tarIqi `adda .hamIrahu wa-huwa rAkibuN, fa-wa^gadahA tis`aTaN.
_tumma nazala wa-`addahA fa-ra'AhA `a^saraTuN fa-qAla: \\\
'am^sI wa-'aksibu .himAraN, 'af.dalu min 'an 'arkaba wa-'a_hsara .himAraN.
\end{arab}
\begin{arab}[trans] % transliteration
\begin{center}\bfseries\large ^gu.hA wa-.hamIruhu al-`a^saraTu\end{center}
i^starY ^gu.hA `a^saraTa .hamIriN.
fari.ha bihA wa-sAqahA 'amAmahu,_tumma rakiba wA.hidaN minhA.
wa-fI al-.t.tarIqi `adda .hamIrahu wa-huwa rAkibuN, fa-wa^gadahA tis`aTaN.
_tumma nazala wa-`addahA fa-ra'AhA `a^saraTuN fa-qAla: \\\
'am^sI wa-'aksibu .himAraN, 'af.dalu min 'an 'arkaba wa-'a_hsara .himAraN.
\end{arab}
```

Exa.
3-3-5

جحا وحميره العشرة

اشترى جحا عشرة حمير. فرح بها وساقها أمامه، ثم ركب واحدا منها. وفي الطريق عدّ حميره وهو راكب، فوجدها تسعة. ثم نزل وعدّها فرأها عشرة فقال: أمشي وأكسب حمارا، أفضل من أن أركب وأخسر حمارا.

جُحَا وَحَمِيرُهُ الْعَشْرَةُ

اِشْتَرَى جُحَا عَشْرَةَ حَمِيرٍ. فَرِحَ بِهَا وَسَاقَهَا أَمَامَهُ، ثُمَّ رَكِبَ وَاحِدًا مِنْهَا. وَفِي الطَّرِيقِ عَدَّ حَمِيرَهُ وَهُوَ رَاكِبٌ، فَوَجَدَهَا تِسْعَةً. ثُمَّ نَزَلَ وَعَدَّهَا فَرَأَاهَا عَشْرَةً فَقَالَ: أَمْشِي وَأَكْسِبْ حِمَارًا، أَفْضَلُ مِنْ أَنْ أُرَكِبَ وَأُخْسَرَ حِمَارًا.

جُحَا وَحَمِيرُهُ الْعَشْرَةُ

اِشْتَرَى جُحَا عَشْرَةَ حَمِيرٍ. فَرِحَ بِهَا وَسَاقَهَا أَمَامَهُ، ثُمَّ رَكِبَ وَاحِدًا مِنْهَا. وَفِي الطَّرِيقِ عَدَّ حَمِيرَهُ وَهُوَ رَاكِبٌ، فَوَجَدَهَا تِسْعَةً. ثُمَّ نَزَلَ وَعَدَّهَا فَرَأَاهَا عَشْرَةً فَقَالَ: أَمْشِي وَأَكْسِبْ حِمَارًا، أَفْضَلُ مِنْ أَنْ أُرَكِبَ وَأُخْسَرَ حِمَارًا.

juḥā wa-ḥamīruhu al-‘asharatu

ishtarā juḥā ‘asharata ḥamīrin. fariḥa bihā wa-sāqahā amāmahu, thumma rakiba wāḥidan minhā. wa-fi al-ṭṭarīqi ‘adda ḥamīrahu wa-huwa rākibun, fa-wajadahā tis‘atan. thumma nazala wa-‘addahā fa-ra’āhā ‘asharatun fa-qāla: amshī wa-aksibu ḥimāran, afḍalu min an arkaba wa-akhsara ḥimāran.

The arabxetex package loads the fontspec package, so that it is easy to select different fonts with Arabic characters. The following example typeset an often-used greeting in various fonts. In the comment line (starting with %), you can see the order in which the Arabic characters are input, i.e., the same as in the Latin transcription with the `\textroman` command. The actual definition of the `\Salam` command shows how the low-level display routines invert the Arabic letters automatically within each word (without T_EX having any control). Indeed, the input sequence of the characters is shown in the commented line, where the character `U+202D` (LRO, for “left-to-right override”) has been prepended before each word to force the characters to be displayed left-to-right. Then, the same greeting is displayed in five different Arabic fonts. Note the use of the `\SCAR` command which defines the script as

Arabic and scales the characters so that their form is more visible.

The most common Arabic language greeting used in both Muslim and Christian cultures means Peace be upon you.

As-SalAmu `Alaykum
السلام عليكم
السلام عليكم
السلام عليكم
السلام عليكم
السلام عليكم

```
\usepackage[no-math]{fontspec}
\usepackage{arabxetex}
\setmainfont{Arial Unicode MS}
\providecommand\SCAR{Script=Arabic,Scale=2.}
\newfontfamily\arSch[\SCAR]{Scheherazade}
\newfontfamily\arTyp[\SCAR]{Arabic Typesetting}
\newfontfamily\arTra[\SCAR]{Traditional Arabic}
\newfontfamily\arTah[\SCAR]{Tahoma}
\newfontfamily\arAri[\SCAR]{Arial Unicode MS}
\let\arabicfont\arSch
%\providecommand\Salam{السلام عليكم}
\providecommand\Salam{السلام عليكم}
The most common Arabic language greeting used
in both Muslim and Christian cultures means
\underline{Peace be upon you}.
\begin{arab}[utf]
\textroman{As-SalAmu `Alaykum}\
{\arSch\Salam}\newline{\arTyp\Salam}\newline
{\arTra\Salam}\newline{\arTah\Salam}\newline
{\arAri\Salam}
\end{arab}
```

Exa.
3-3-6

The following example shows how easy it is to include \LaTeX commands inside Arabic text. For the Arabic source (at the right) each word has been preceded by the LRO (U+202D, as explained for Example 3-3-6) character to show the order (left-to-right) in which the Arabic characters are input. Note how the `flushleft` environment typesets the Arabic text effectively `flushright`.

<p>دنيا جي پيدائش</p> <p>۱ شروعات ۾ خدا زمين ۽ آسمان کي پيدا ڪيو.</p> <p>۲ ان وقت زمين بي ترتيب ۽ ويران هئي. اونهي سمنڊ جو مٿاڇرو اوندهه سان ڍڪيل هو ۽ پاڻي جي مٿان خدا جي روح ڦيرا پئي ڪي</p> <p>۳ تڏهن خدا حڪم ڏنو ته ”روشني ٿي.“ سو روشني ٿي پيئي.</p>	<pre>\color[rgb]{0,0,1} \begin{arab}[utf] \begin{center} ش \[3mm] \end{center} \begin{flushleft} \fbbox{1} نېمڙ ادخ ۾ تناغوروش \fbbox{2} ويڪ ادپي پيڪ نامس آ \fbbox{3} بيترتبي نيمز تقو نا نممس يهنا. يئ ه نارپو وه لپيڪ ناس ههنا ورتھاتم جو ادخ ناتم يچ گھڻاپ \fbbox{4} يڪ يئ پ اري ٿ حور جي ينشور ”ه ونه مڪح ادخ نهنت يئ يپ ي ٿ ينشور وس.“ يئ ٿ \end{flushleft} \end{arab}</pre>
---	--

Contextual analysis of hamza

Our next example is from the $\text{ArabX}\TeX$ manual. As with `arabxetex`, a contextual analysis of the input encoding is performed (at the font-mapping level) to automatically determine the carrier of the *hamza*,

as illustrated next.

```
\usepackage{arabxetex}
\newfontfamily\arabicfont[Script=Arabic,Scale=1.0]{Scheherazade}
\begin{arab}[voc]
'amruN, 'ibiluN, 'u_htuN, '"u_ht"uN, '"Uql"Id"Is, ra'suN, 'ar'asu,
sa'ala, qara'a, bu'suN, 'ab'usuN, ra'ufo, ru'asA'u, bi'ruN, 'as'ilaTuN,
ka'iba, qA'imuN, ri'AsaTuN, su'ila, samA'uN, barI'uN, sU'uN, bad'uN,
^say'uN, ^say'iN, ^say'aN, sA'ala, mas'alaTuN, saw'aTuN, _ha.tI'aTuN,
jA'a, ridA'uN, ridA'aN, jI'a, radI'iN, sU'uN, .daw'uN, qay'iN, .zim'aN
, yatasA'alUna, 'a`dA'akum, 'a`dA'ikum, 'a`dA'ukum maqrU'aT, mU'ibAt,
taw'am, yas'alu, 'a.sdiq^A$; $'uh_u, ya^g^I'u, s^U'ila
\end{arab}
```

Exa.
3-3-7

أمر، إيل، أخت، أخت، أوقلديس، رأس، رأس، أرأس، نأل، قرأ، بوس، أبوس، رؤف، رؤساء، بئر، أسئلة، كيب، قائم، رأسة، شيل، سماء، بري، شوء، بدء،
شيء، شينا، سائل، مسألة، سواة، خطيئة، جاء، رداء، رداء، جين، ردي، شوء، ضوء، في، ظمنا، يتساءلون، أعداءكم، أعداءكم، أعداءكم مقروؤة،
موتيات، توأم، يسأل، أصدقاؤه، يجيء، سويل

Typesetting the Qur'ān

As the Holy Qur'ān (القرآن الكريم) plays an important role in Islamic culture, its high-quality typesetting is an important and rather complex task, and typeset examples of that book by professional typesetters are often real works of art. Nowadays several OpenType fonts cover the full Unicode character range for the Arabic script, and it is possible to achieve quite acceptable results. The following example from the ArabTeX manual, which uses the fonts *Scheherazade* shows some typographic features which characterize typical printed editions from Saudi Arabia.

Note in particular the definition of the *hamza* placed directly over the baseline instead as over the *alif*, something that is usually not encoded in a Unicode font, but it is easily emulated by a TeX macro (`\hamzaB`).

```
\usepackage{arabxetex}
\newfontfamily\arabicfont[Script=Arabic,Scale=1.0]{Scheherazade}
\newcommand{\hamzaB}{\char"200D\char"0640\raisebox{-0.95ex}{\char"0654}\char"200D}
\begin{arab}[fullvoc]
mina 'l-qur'Ani 'l-karImi, sUraTu 'l-ssajdaTi 15--16:\\
'innamA yu'minu bi-\hamzaB a|"Ay__atInA 'lla_dIna 'i_dA _dukkirUA bihA
_harrUA sujjadaN wa-sabba.hUA bi-.hamdi rabbihim wa-hum lA yastakbirUna
SAJDA [[15]] tatajAfY_a junUbuhum `ani 'l-ma.dAji`i yad`Una rabbahum
_hawfaN wa-.tama`aN wa-mimma razaqn_ahum yunfiqUna [[16]]\\
sUraTu 'l-baqaraTi 71--72:\\
qAla 'innahu, yaqUlu 'innahA baqaraTuN lla _dalUluN tu_tIru 'l-'ar.da wa-lA
tasq.I 'l-.har_ta musallamaTuN lla ^siyaTa fIhA|^JIM qAluW" 'l-\hamzaB a__ana
ji'ta bi-'l-.haqqi|^JIM fa_daba.hUha wa-mA kAdduW" yaf`alUna [[71]] wa-'i_d
qatalum nafsaN fa-udda$, $_ara|'|_i"tum fIhA|^SLY wa-al-ll_ahu mu_hrijuN mma
kun"tum taktumUna [[72]]
\end{arab}
```

Exa.
3-3-8

مِنَ الْقُرْآنِ الْكَرِيمِ، سُورَةُ السَّجْدَةِ ١٥-١٦:
إِنَّمَا يُؤْمِنُ بِآيَاتِنَا الَّذِينَ إِذَا ذُكِرُوا بِهَا خَرُّوا سُجَّدًا وَسَبَّحُوا بِحَمْدِ رَبِّهِمْ وَهُمْ لَا يَسْتَكْبِرُونَ ﴿١٥﴾ تَتَجَافَى جُنُوبُهُمْ عَنِ الْمَضَاجِعِ يَدْعُونَ رَبَّهُمْ خَوْفًا وَطَمَعًا وَمِمَّا
رَزَقْنَاهُمْ يُنفِقُونَ ﴿١٦﴾
سُورَةُ الْبَقَرَةِ ٧١-٧٢:
قَالَ إِنَّهُ، يَقُولُ إِنَّهَا بَقَرَةٌ لَا ذَلُولَ تُبِيرُ الْأَرْضَ وَلَا تَسْقِي الْحَرْثَ مُسَلِّمَةٌ لَا يَبَيْتُ فِيهَا قَالُوا لَنْ نَجِدَ بِالْحَقِّ فَدْبُحُوهَا وَمَا كَادُوا يَفْعَلُونَ ﴿٧١﴾ وَإِذْ قَتَلْتُمْ نَفْسًا فَاذْرَأْتُمْ
فِيهَا وَاللَّهُ مُخْرِجٌ مِمَّا كُنْتُمْ تَكْتُمُونَ ﴿٧٢﴾

The following example is a table from a grammar book showing prefix and suffix constructs for Arabic verbs. It is seen how easy it is to mix the Latin and Arabic alphabets and use a large set of L^AT_EX commands. We only show the beginning of the source file. As default Arabic font we select *Traditional Arabic*. Note how we introduce the Arabic environment `arab` in the preamble for the third, fifth, and seventh columns (the `[utf]` option is implicit, since not needed).

```
% from http://en.wikipedia.org/wiki/Arabic_grammar
\documentclass[a4paper]{article}
\usepackage[no-math]{fontspec}
\usepackage{array}
\usepackage{arabxetex}
\setmainfont{Minion Pro}
\newfontfamily\arabicfont[Script=Arabic,Scale=1.2]{Traditional Arabic}
\begin{document}
\begin{tabular}{@l*3l}>\begin{arab}}r<\end{arab}}@{}
\multicolumn{7}{c}{Prefixes and suffixes of the Arabic verb}\\
& \multicolumn{2}{c}{Perfective}
& \multicolumn{2}{c}{Imperfective}
& \multicolumn{2}{c}{Subjunctive and Jussive}\\
\multicolumn{7}{c}{\textbf{Singular}} \\
3rd (m.)
& STEM\textbf{-a} & & & & &
& \textbf{بَتَكَ} & & & & &
& \textbf{بَتَكَ} & & & & &
& \multicolumn{2}{c}{\emph{no written change}}\\
3rd (f.)
& STEM\textbf{-at} & & & & &
& \textbf{بَتَكَتْ} & & & & &
& \textbf{بَتَكَتْ} & & & & &
& \multicolumn{2}{c}{\emph{no written change}}\\
```

Prefixes and suffixes of the Arabic verb						
	Perfective		Imperfective		Subjunctive and Jussive	
			Singular			
3rd (m.)	STEM-a	كَتَبَ	ya-STEM	يَكْتُبُ	no written change	
3rd (f.)	STEM-at	كَتَبَتْ	ta-STEM	تَكْتُبُ	no written change	
2nd (m.)	STEM-ta	كَتَبْتَ	ta-STEM	تَكْتُبُ	no written change	
2nd (f.)	STEM-ti	كَتَبْتِ	ta-STEM-īna	تَكْتُبِينَ	ta-STEM-ī	تَكْتُبِي
1st	STEM-tu	كَتَبْتُ	a-STEM	أَكْتُبُ	no written change	
			Dual			
3rd (m.)	STEM-ā	كَتَبَا	ya-STEM-āni	يَكْتُبَانِ	ya-STEM-ā	يَكْتُبَا
3rd (f.)	STEM-atā	كَتَبْتَا	ta-STEM-āni	تَكْتُبَانِ	ta-STEM-ā	تَكْتُبَا
2nd (m. & f.)	STEM-tumā	كَتَبْتُمَا	ta-STEM-āni	تَكْتُبَانِ	ta-STEM-ā	تَكْتُبَا
			Plural			
3rd (m.)	STEM-ū	كَتَبُوا	ya-STEM-ūna	يَكْتُبُونَ	ya-STEM-ū	يَكْتُبُوا
3rd (f.)	STEM-na	كَتَبْنَ	ya-STEM-na	يَكْتُبْنَ	no written change	
2nd (m.)	STEM-tum	كَتَبْتُمْ	ta-STEM-ūna	تَكْتُبُونَ	ta-STEM-ū	تَكْتُبُوا
2nd (f.)	STEM-tunna	كَتَبْتُنَّ	ta-STEM-na	تَكْتُبْنَ	no written change	
1st	STEM-nā	كَتَبْنَا	na-STEM	نَكْتُبُ	no written change	

Another grammatical table showing derivations from sound verbs is our next example, where we

Exa.
3-3-9

use *Arabic Typesetting* font.

Exa.
3-3-10

Sound verbs (3rd sg. masc.)				
	Active voice		Passive voice	
	<i>Past</i>	<i>Present</i>	<i>Past</i>	<i>Present</i>
I	فَعَلَ	يَفْعَلُ	فُعِلَ	يُفْعَلُ
II	فَعَّلَ	يَفْعِلُ	فُعِّلَ	يُفْعَلُ
III	فَاعَلَ	يُفَاعِلُ	فُوعِلَ	يُفَاعَلُ
IV	أَفْعَلَ	يُفْعِلُ	أُفْعِلَ	يُفْعَلُ
V	تَفَعَّلَ	يَتَفَعَّلُ	تُفَعَّلَ	يَتَفَعَّلُ
VI	تَفَاعَلَ	يَتَفَاعَلُ	تُفَوِّعَلُ	يَتَفَاعَلُ
VII	اِنْفَعَلَ	يَنْفَعِلُ	<i>not available</i>	
VIII	اِفْتَعَلَ	يَفْتَعِلُ	اُفْتَعِلَ	يَفْتَعَلُ
IX	اِفْعَلَ	يَفْعَلُ	<i>not available</i>	
X	اِسْتَفَعَلَ	يَسْتَفَعِلُ	اُسْتَفَعِلَ	يُسْتَفَعَلُ

```

\usepackage[no-math]{fontspec}
\usepackage{array}
\usepackage{arabxetex}
\setmainfont{Minion Pro}
\newfontfamily\arabicfont[Script=Arabic,Scale=1.2]
{Arabic Typesetting}
\begin{tabular}{@{}c*4>{\begin{arab}[voc]}r<{\end{arab}}@{}
\multicolumn{5}{c}{\textbf{Sound verbs} (3rd sg. masc.)}
\\
& \multicolumn{2}{c}{\textbf{Active voice}}
& \multicolumn{2}{c}{\textbf{Passive voice}}
\\
& \multicolumn{1}{c}{\emph{Past}}
& \multicolumn{1}{c}{\emph{Present}}
& \multicolumn{1}{c}{\emph{Past}}
& \multicolumn{1}{c}{\emph{Present}}
\\
\textbf{I} & & \textbf{II} & & \textbf{III} & & \textbf{IV} & & \textbf{V} & & \textbf{VI} & & \textbf{VII} & & \textbf{VIII} & & \textbf{IX} & & \textbf{X}
\end{tabular}

```

We can even get more fancy and specify all Arabic characters on input by their Unicode code position (this is often used on the Web with the character reference syntax `&xxxx;`, where `xxxx` is the code position). The following table of countries in the Arab world is taken from the Web site indicated below (only the first part of the source is shown). The *Arial Unicode MS* font is used for most of the Arabic, except for the right-hand column in the table, for which *Old Antic Bold* has been selected. Note the order of typesetting of the columns in this table (from right to left). In fact, in English this table would have the following structure:

	country	capital	people
North Africa	Tunesia	Tunis	Tunesians
	Algeria	Algiers	Algerians
	...		

For the Arabic version shown below, these columns have to be mirrored by hand from left to right by specifying the “people” columns entries first, then the “capital” column entries, etc.

```

% from http://www.arabiyya.123.fr/spip/spip.php?article13
\documentclass[a4paper]{article}
\usepackage[no-math]{fontspec}
\usepackage{array}
\usepackage{arabxetex}
\setmainfont{Minion Pro}
\newfontfamily\arabicfont[Script=Arabic,Scale=1.0]{Arial Unicode MS}
\newfontfamily\Antic[Script=Arabic,Scale=1.2]{Old Antic Bold}

```

```

\begin{document}
\begin{arab}
\renewcommand{\arraystretch}{1.1}
\setlength{\extrarowheight}{1mm}
\begin{tabular}{@{}>\Antic}l@{\quad}rrr@{}}
& \char1575\char1604\char1588\char1614\char1593\char1618\char1576
& \char1575\char1604\char1593\char1575\char1589\char1616\char1605\char1577
& \char1575\char1604\char1576\char1614\char1604\char1614\char1583
& \char1576\char1604\char1583\char1575\char1606 \char1580\char1575\char1605\char1593\char1577
& \char1575\char1604\char1583\char1608\char1604 \char1575\char1604\char1593\char1585\char1576\char1610\char1577
& \char1578\char1608\char1606\char1616\char1587\char1610\char1617
& \char1578\char1600\char1615\char1608\char1606\char1616\char1587
& \char1578\char1600\char1615\char1608\char1606\char1616\char1587
& \char1580\char1614\char1586\char1575\char1574\char1616\char1585\char1610\char1617
& \char1575\char1604\char1580\char1614\char1586\char1575\char1574\char1616\char1585
& \char1575\char1604\char1580\char1614\char1586\char1575\char1574\char1616\char1585
\end{tabular}

```

Exa.
3-3-11

الشَّعْبُ	العاصمة	البلد	
تُونِسِيّ	تُونِس	تُونِس	جامعة الصّوالمعربية
جَزَائِرِيّ	الجَزَائِر	الجَزَائِر	
لِيبِيّ	طَرَابُلس	لِيبِيَا	
مَغْرِبِيّ	الرباط	المَغْرِب	
موريتانيّ	نواكشوط	موريتانيا	
سودانيّ	الخَرْطوم	السُّودان	وادي النيل
مِصْرِيّ	القاهرة	مِصْر	
جيبوتيّ	جيبوتي	جيبوتيا	القرن الإفريقي
صوماليّ	مَقْدِيشو	الصُّومال	
أردنيّ	عمان	الأردن	الغولاصيب
فِلَسْطِينِيّ	رامالله	فِلَسْطِين	
سُورِيّ	دِمَشق	سُورِيَا	
عِرَاقِيّ	بَغْداد	العِرَاق	
لِبنَانِيّ	بِيرُوت	لِبنَان	
إِمَارَاتِيّ	أبوظبِيّ	الإمارات العربية المتّحدة	الجزيرة العربية
بَحْرِينِيّ	مَنَامَة	البَحْرِين	
سُعودِيّ	الرياض	العربيّة السُّعوديّة	
عُمَانِيّ	مَسْقَط	عُمَان	
قَطْرِيّ	الدوحة	قَطْر	
كُوَيْتِيّ	الكُوَيْت	الكُوَيْت	
يَمِينِيّ	صَنَعَاء	اليَمِن	
قَمْرِيّ	مارونيّ	جُزُر القَمَر	جُزُر القَمَر

3.3.2.1 ArabX₂TeX: typesetting Persian

The following is an example from the ArabTeX manual.

```
\usepackage{arabxetex}
\newfontfamily\arabicfont[Script=Arabic,Scale=1.0]{Scheherazade}
\newfontfamily\farsifont[Script=Arabic,Scale=1.1]{Farsi Simple Bold}
\begin{farsi}[voc]
_hwAb, xwI^s, _hwod, ^ceH, naH, yal_aH, _hAneH, _hAneHhA, _hAneH-hA,
ketAb-e, U, rAh-e, t_U, nAmeH-i, man, bInI-e, An, mard, pA-i, In,
zan, bAzU-i, In, zan, dAr-_i, man, _hU-_i, t_U, nAmeH-_i, sormeH-_i,
gofteH-_i, ketAb-I, rAh-I, nAmeH-I, dAnA-I, pArU-I, dAnA-I-keH,
pArU-I-keH, rafteH-am, rafteH-Im, AnjA-st, U-st, t_U-st, ketAb-I-st,
be-man, be-t_U, be-An, be-In, be-insAn, beU, be-U, .sA.heb"|_hAneH,
pas"|andAz, naw"|AmUz
\end{farsi}
```

Exa.
3-3-12

خواب، خویش، خُودِ چہ، نہ، یکہ، خانہ، خانہہا، خانہہا، کتاب، او، راہ، تو، نامید، من، بینی، آن، مرد، پای، این، زن، باژوی،
 این، زن، دای من، خوی، تو، نامید، سرمہ، گفتید، کتابی، راہی، نامہ ای، داناہی، پاوی، داناہیکہ، پاویکہ، رفتہ ام، رفتہ ایم،
 آنجاست، اوست، توست، کتابیست، بمن، بتو، بان، پلین، باتسان، باو، باو، صاحب خانہ، پس ندان نواموز

3.3.2.2 ArabX₂TeX: Various ways of typesetting Urdu

Like Persian (Farsi), Urdu is an Indo-European language written in the Arabic alphabet (see <http://en.wikipedia.org/wiki/Urdu>). However, Urdu letters (and their fonts) have forms that are quite different from their “common” Arabic equivalents as the next short example shows.¹ We first use an undifferentiated “Arabic” font (*Code2000*).

```
\usepackage{arabxetex}
\newfontfamily\arabicfont[Script=Arabic,Scale=1.0]{Scheherazade}
\newfontfamily\urdufont[Script=Arabic,Scale=1.1]{Code2000}
\begin{urdu}[novoc]
,ham `i^sq kE mArO.n kA itnA ,hI fasAna,h ,hae\\
rOnE kO na,hI.n kO'I ,ha.nsnE kO zamAna,h ,hae\par
ya,h kiskA ta.sawwur ,hae ya,h kiskA fasAna,h ,hae\\
jO a^sk ,hae A.nkhO.n mE.n tasbI.h kA dAnA ,hae
\end{urdu}
```

Exa.
3-3-13

ہم عشق کے ماروں کا اتنا ہی فسانہ ہے
 رونے کو نہیں کیوٹی ہوسنے کو زمانہ ہے
 یہ کسکا تصور ہے یہ کسکا فسانہ ہے
 جو اشک ہے آنکھوں میں تسبیح کا دانا ہے

Then we show the same example with two other fonts which have been designed to show Urdu variant of typesetting the letters. The example also shows that it is enough to change the definition of the `\urdufont` command to contain the OpenType name of the font one actually want to use.

¹The text is borrowed from <http://tabish.freeshell.org/u-trans/urducode.html>, a short page on ArabTeX coding for Urdu.

ہم عشق کے ماروں کا اتنا ہی فسانہ ہے
 رونے کو نہیں کوئی ہنسنے کو زمانہ ہے
 یہ کسکا تصور ہے یہ کسکا فسانہ ہے
 جواشک ہے آنکھوں میں تسبیح کا دانا ہے

Nafees Pakistani Naskh

The web page referenced in the footnote 1 refers to the Urdu font *Urdu Nastaliq Unicode*, which comes with a few examples, one of which is a ghazal.¹ We use it to show the difference in typesetting of the Urdu text with a “global” font for Arabic characters (*Arial Unicode MS*), seen at the left, and *Urdu Nastaliq Unicode*, specifically developed for typesetting Urdu texts, seen at the right.

ہم عشق - ماروں اتنا ہی فسانہ ہے
 رونے و نہیں وئی ہنسنے و زمانہ ہے
 یہ - اتصو رہے یہ - افسانہ ہے
 جوائے ہے آز ہوں میں تسبیح ادا نا ہے

Nafees Riqā

غَزَل

شامِ غَم کے اسپر ہیں ہم لوگ
 سُبْحِ نَو کے سفیر ہیں ہم لوگ
 بُجھ چُکائے چراغِ گودِ ل کا
 پھر بھی روشن ضمیر ہیں ہم لوگ
 یاس و غم کی بے گر کوئی قیمت
 پھر تو سب سے امیر ہیں ہم لوگ
 ایک مَوہوم سا تصوّر رہیں
 ایک مدّھم لگیں ہیں ہم لوگ
 قاتلوں کے نگر میں ائے بارو
 اہل دل کے مُشیر ہیں ہم لوگ
 ایک نظر کی ہمیں بھی دے دو بھیک
 راہ چلتے فقیر ہیں ہم لوگ
 پھر ملیگا نہ سادہ دل ہم سا
 فی زمانہ نریر ہیں ہم لوگ
 خود ہمیں بھی ہے آسرا در کار
 مت کہو دستگیر ہیں ہم لوگ
 ز فرامر

غَزَل

شامِ غَم کے اسپر ہیں ہم لوگ
 سُبْحِ نَو کے سفیر ہیں ہم لوگ
 بُجھ چُکائے چراغِ گودِ ل کا
 پھر بھی روشن ضمیر ہیں ہم لوگ
 یاس و غم کی بے گر کوئی قیمت
 پھر تو سب سے امیر ہیں ہم لوگ
 ایک مَوہوم سا تصوّر رہیں
 ایک مدّھم لگیں ہیں ہم لوگ
 قاتلوں کے نگر میں ائے بارو
 اہل دل کے مُشیر ہیں ہم لوگ
 ایک نظر کی ہمیں بھی دے دو بھیک
 راہ چلتے فقیر ہیں ہم لوگ
 پھر ملیگا نہ سادہ دل ہم سا
 فی زمانہ نریر ہیں ہم لوگ
 خود ہمیں بھی ہے آسرا در کار
 مت کہو دستگیر ہیں ہم لوگ
 ز فرامر

Exa.
3-3-14

¹The ghazal is a poetic form consisting of couplets which share a rhyme and a refrain. Each line must share the same meter. Ghazals are traditionally expressions of love, separation and loneliness, a poetic expression of both the pain of loss or separation and the beauty of love in spite of that pain. The form is ancient, originating in 10th century Persian verse. It is considered by many to be one of the principal poetic forms the Persian civilization offered to the eastern Islamic world, see <http://en.wikipedia.org/wiki/Ghazal>. Nowadays the ghazal is most prominently a form of Urdu poetry, see <http://www.urdupoetry.com>.

3.3.3 Arabic presentation forms

The preferred Unicode block for the Arabic scripts is “Arabic” (U+0600–U+06FF), which is complemented by the “Arabic Supplement” block (U+0750–U+077F), which adds letters mainly used in Northern and Western African languages.

Languages written in the Arabic script have often a long tradition of cursive handwriting on manuscripts. In particular, Arabic itself is closely linked to the spread of the Koran and, more generally, Islamic culture. Therefore letter sequences, or even words have presentations that are different from the linear combination of the composing letters. Moreover, these forms often depend on the language. Therefore Unicode contains an “Arabic Presentation Forms-A” block (U+FB50–U+FDFF). This is subdivided into several parts: glyphs for contextual forms of letters for Persian, Urdu, Sindhi, etc. (U+FB50–U+FBF1), glyphs for contextual forms of letters for Central Asian languages (U+FBF2–U+FBF9), ligatures (two elements, U+FBFA–U+FBFD), punctuation (U+FBFE–U+FBFF), ligatures (three elements, U+FD50–U+FD7F), Noncharacters (U+FD80–U+FD9F), word ligatures (U+FDFA–U+FDFF), currency sign (U+FDFF), and a symbol (U+FDFF).

There is also an “Arabic Presentation Forms-B” block (U+FE70–U+FEFF), which contains mainly contextual shape variations that are important semantically for Arabic mathematics: glyphs for spacing forms of Arabic points (U+FE70–U+FE7F), and basic glyphs for Arabic language contextual forms (U+FE80–U+FEFF).

One example is U+FDFA (Arabic ligature *Allah* isolated form), whose support in various fonts is shown here. The issue of typesetting the name of God in Arabic, which is quite complex, is explained in detail in the ArabX₃TeX manual.

Fonts from or licensed to Microsoft:

Times New Roman الله — Arial الله — Courier New لله — Microsoft Sans Serif الله — Arial Unicode MS الله — Arabic Transparent لله — Simplified Arabic لله — Simplified Arabic Fixed لله — WinSoft Serif Pro Medium لله — Traditional Arabic لله — Arabic Typesetting الله — Old Antic Bold الله — Farsi Simple Bold لله

Urdu: Nastaleeq Like لله PakType Naqsh الله, which contains also presentation forms for the following Arabic ligatures:

U+FDFA (SALLALLAHOU ALAYNE WASALLAM)

U+FDFF (JALLAJALALOUHOU)

Adobe (<http://www.adobe.com>): Adobe Arabic الله

SIL (www.sil.org): Scheherazade الله — Lateef لله

Arabeyes (www.arabeyes.org): KacstBook لله KacstFarsi لله

Overview of all input conventions

Table 3.3 shows all complete list of all input character combinations used by arabxetex. The input sequences are ordered alphabetically following the most significant letter of the ASCII input code. The characters are accompanied by their (hexadecimal) Unicode number. The following color conventions are used: *red* means that the glyph is the default for the given input code, and that it is available in all languages except those where different glyphs are shown (in *black*). That default glyph is also displayed in *light gray* under each language in which it is featured. Glyphs in *blue* are archaic forms (e.g., old Urdu). An asterisk after the Unicode number means that the character was not available with arabxetex. *Green* glyphs are special: either they are used to represent defective writing or they provide characters for other languages. Those shown in the column for Arabic are available by default.

Table 3.3: All arabxetex input conventions

code	arab	farsi	urdu	pashto	sindhi	ottoman	kurdish	kashmiri	malay	uighur
a	ﺍ 064E						ﺍ/ﺍ			
A	ﺍ 0627						ﺍ/ﺍ			
.a								ﺍ 0654		
.A								ﺍ 0672		
_a	ﺍ 0670									
_A	ﺍ 0670									
:a										ﺍ
b	ﺏ 0628	ﺏ	ﺏ	ﺏ	ﺏ	ﺏ	ﺏ	ﺏ	ﺏ	ﺏ
B	ﺏ 0640									
.b	ﺏ 066E									
:b					ﺏ 067B					
bh					ﺏ 0680					
c					ﺝ 0681				ﺝ 0686	
,c				ﺝ 0685		ﺝ 062C		ﺝ 0686		
^c	ﺝ 0686	ﺝ 0686	ﺝ	ﺝ	ﺝ	ﺝ 0686	ﺝ	ﺝ	ﺝ	ﺝ
^ch					ﺝ 0687					
:c				ﺝ 0682*						
.^c	ﺝ 06BF*									
d	ﺩ 062F	ﺩ	ﺩ	ﺩ	ﺩ	ﺩ	ﺩ	ﺩ	ﺩ	ﺩ
.d	ﺩ 0636	ﺩ	ﺩ	ﺩ	ﺩ	ﺩ	ﺩ	ﺩ	ﺩ	ﺩ
,d			ﺩ 0688	ﺩ 0689	ﺩ 068A					
.,d			ﺩ 068B*							
^d	ﺩ 06EE*				ﺩ 068E*					
_d	ﺩ 0630	ﺩ	ﺩ	ﺩ	ﺩ	ﺩ	ﺩ	ﺩ	ﺩ	ﺩ
:d			ﺩ 0690*							
dh					ﺩ 068C					
,dh					ﺩ 068D					
e		ﻩ	ﻩ	ﻩ 0659	ﻩ 06D0		ﻩ ﻩ	ﻩ 06D2+0658		ﻩ 06D0
E		ﻩ	ﻩ 06D2	ﻩ 06D0	ﻩ 06D0		ﻩ ﻩ	ﻩ 06D2		ﻩ 06D0
ee				ﻩ 06CD	ﻩ					
ae	ﻩ				ﻩ					
Ee					ﻩ 06CD					
_e		ﻩ								
'e							ﻩ			
'E			ﻩ 06D3							
f	ﻑ 0641	ﻑ	ﻑ	ﻑ	ﻑ	ﻑ	ﻑ	ﻑ	ﻑ	ﻑ
.f	ﻑ 06A1									
g		ﻎ 06AF	ﻎ	ﻎ 06AB	ﻎ	ﻎ	ﻎ	ﻎ	ﻎ 0762	ﻎ
G				ﻎ 06AB	ﻎ	ﻎ	ﻎ	ﻎ		
.g	ﻎ 063A	ﻎ	ﻎ	ﻎ	ﻎ	ﻎ	ﻎ	ﻎ	ﻎ	ﻎ
:g					ﻎ 06B3					
,g	ﻎ 06AC				ﻎ 06B4*					
^g	ﻎ 062C	ﻎ	ﻎ	ﻎ	ﻎ	ﻎ 06A0	ﻎ 063A	ﻎ	ﻎ 06A0	ﻎ 063A

code	arab	farsi	urdu	pashto	sindhi	ottoman	kurdish	kashmiri	malay	uighur
gh					گھ					
h	ه 0647	ه	ه 06BE	ه	ه	ه	ه	ه	ه	ه
H		ه 0647	ه 06C3	ه 0647						
.h	ح 062D	ح	ح 06C1	ح	ح	ح	ح	ح	ح	
,h								ه		
_h	ح 062E	ح	ح	ح	ح	ح	ح	ح	ح	
i	ی 0650									ی 066E
I	ی 064A	ی 06CC	ی	ی	ی	ی	ی نی ای	ی	ی	
.I	ی 06CC*									
_i	ی 0656									
j	ج 062C	ج	ج	ج	ج	ژ 0698	ژ 0698	ج	ج	ج
:j					ج 0684					
jh					چ 06A9					
k	ك 0643	ك	ك	ك	ك 06AA	ك	ك	ك	ك	ك
.k	ك 06A9					ق 0642				
_k	ك 063A									
kh					ک					
l	ل 0644	ل	ل	ل	ل	ل	ل 06B6*	ل	ل	ل
.l							ل 06B5			
^l										
m	م 0645	م	م	م	م	م	م	م	م	م
.mIN					م 06FE					
'iIN					م 06FD					
n	ن 0646	ن	ن	ن	ن	ن	ن	ن	ن	ن
aN	ا 064B									
uN	ا 064C									
iN	ا 064D									
.n			ن 06BA							
..n					ن 06B2*					
,n				ن 06BC	ن 06BB					
^n					ن 06B3			ن 06BD		ن 06AD
:n					ن 06B1					
o		و	و	و 0657			و ئو ئو	و 06C6		و
O		و	و	و				و 06C4		
ao								و وآ		
.o										
.O										
_o		و								
_O		و								
:o										و ئو 06C6
:O	و 06FC									
p	پ 067E	پ	پ	پ	پ	پ	پ	پ	پ 06A8	پ
ph					پ 06A6					
q	ق 0642	ق	ق	ق	ق	ق	ق	ق	ق	ق
.q	ق 066F									

code	arab	farsi	urdu	pashto	sindhi	ottoman	kurdish	kashmiri	malay	uighur
r	ر 0631	ر	ر	ر	ر	ر	ر	ر	ر	ر
.r	ر 0694 ^a						ر 0695			
,r			ر 0691	ر 0693	ر 0699		ر 0694 ^a			
^r	ر 065F ^c						ر 0692 ^a			
:r	ر 0697 ^c									
s	س 0633	س	س	س	س	س	س	س	س	س
.s	س 0635	س	س	س 069A	س	س 0634	س	س	س	
,s				س 069A		س 0634				
^s	س 0634	س	س	س	س 062B	س	س	س	س	س
_s					س 062B					
:s	س 069B									
t	ت 062A	ت	ت	ت	ت	ت	ت	ت	ت	ت
T	ت 062A							ة		
.t	ت 0637	ط	ط	ط	ط	ط	ط	ط	ط	
,t			ت 0679	ت 067C	ت 067D					
_t	ت 062B	ت	ت		ت	ت	ت	ت	ت	
th					ت 067F					
,th					ت 067A					
u	و 064F	و	و	و	و	و	ئولو ئولوو	-	-	ئولو 06C7
U	و 0648	و	و	و	و	و	ئولو ئولوو	ئولو 0648+0657	و	
.u								ئولو 0655		
.U								ئولو 0673		
_u	و 0657									
:u										ئولو 06C8
:U	و 06C7									
v	و 06A4								و 06CF	
w	و 0648	و	و	و	و	و	و	و	و	و 06CB
W	و 0648									
^w	و 06C9 ^a									
:w	و 06CA ^a									
x	خ 062E	خ	خ	خ	خ	خ	خ	خ	خ	خ
Y	ي 064A	ي 06CC	ي	ي	ي	ي	ي	ي	ي	ي
Y	ي 0649									
.Y								ي		
Z	ز 0632	ز	ز	ز	ز	ز	ز	ز	ز	ز
.Z	ظ 063B	ظ	ظ	ظ	ظ	ظ	ظ	ظ	ظ	
,Z				ز 0696		ظ 0636				
^Z		ز 0698	ز	ز				ز		ز
_Z	ز 0630									
:Z								ظ 0636		
'	ع 0621	ع	ع	ع	ع	ع	ع	ع	ع	ع
`	ع 0639	ع	ع	ع	ع	ع	ع	ع	ع	ع

^a For Western Punjabi (Lahnda).

^b Alternative form of و in Malay.

^c For Dargwa (language of Dagestan).

^d For Kirgiz (and Uighur).

^e To transliterate dialects and foreign words.

^f Alternative to د.

Maghribi Arabic is identical to Arabic except for the three letters ڤ, ڨ and ڰ which yield the glyphs ڤ (U+06A2), ڨ (U+06A7), and ڰ (U+06A5), respectively.

3.4 Typesetting Chinese

Ideographics CJK (Chinese, Japanese, Korean) scripts can be handled by $X_{\text{E}}\text{TeX}$ by directly using the corresponding Unicode characters in the input stream. The following example shows a few Kanji characters and their pronunciation. Note the use of the *Color* argument on the `\font` command (see Section 2.3 for details of $X_{\text{E}}\text{TeX}$'s extensions to TeX 's standard `\font` command).

<code>\font\han="STSong;color=660000" at 12pt</code>	書
<code>\font\rom="Gentium;color=006600" at 8pt</code>	ka-ku
<code>\newcommand\hc[2]{\begin{tabular}{l}</code>	最
<code> \han #1\[-1mm]\rom #2\end{tabular}}</code>	も
<code>\begin{tabular}{l}</code>	motto-mo
<code> \hc{書}{ka-ku}</code>	最
<code> \hc{最}{motto-mo}</code>	後
<code> \hc{最後}{sai-go}</code>	sai-go
<code> \hc{働}{hatara-ku}</code>	働
<code> \hc{海}{umi}</code>	hatara-ku
<code>\end{tabular}</code>	海
	umi

By default, $X_{\text{E}}\text{TeX}$ does not handle some important aspects of Chinese typesetting, such as automatic font switching between Chinese and Western characters, skip adjustments for fullwidth punctuations, or automatic skip insertions between Chinese and Western characters or math formulas.

3.4.1 The xeCJK Package

Wenchang Sun developed the `xecjk` package to help $X_{\text{E}}\text{TeX}$ users typeset texts based on CJK scripts more easily. The `xeCJK` package offers the following main features.

1. initializes different default fonts for CJK and other scripts;
2. spaces are automatically ignored between CJK characters;
3. supports several CJK punctuation processing modes;
4. can adjust the space between CJK and other characters automatically.

Note that `xeCJK` needs version 0.9995.0 of $X_{\text{E}}\text{TeX}$ or a later version.

3.4.1.1 Usage

```
\usepackage[Options]{xeCJK}
```

The *options* are the following.

BoldFont Create “synthetic bold” fonts for CJK characters. Will be overridden by specifying *BoldFont* in the definition of a CJK family.

SlantFont Create slanted fonts for CJK characters. Will be overridden by specifying *ItalicFont* in the definition of a CJK family.

CJKnumber Load the `CJKnumb` package.

CJKaddspaces Add spaces between CJK and other characters if there is none.

CJKnormalspaces Ignore only spaces between CJK characters and leave spaces between CJK and other characters untouched.

CJKchecksingle Avoid that a single Chinese character monopolizes a line.

```
\setCJKmainfont[<font features>]{font name}
\setCJKsansfont[<font features>]{font name}
\setCJKmonofont[<font features>]{font name}
```

These three commands, which are analogues of `\setmainfont`, `\setsansfont`, and `\setmonofont`, respectively, set different default fonts for CJK characters only, without affecting other scripts.

When in the definition of a CJK typeface the `ItalicFont= {...}` option specified an explicit fontname, then the `SlantFont` option will have no effect for this typeface. Similarly specifying an explicit bold font with `BoldFont= {...}` in the font feature part suppresses the effect of the global `BoldFont` option.

```
\setCJKfamilyfont{familyname}[<font features>]{font name}
```

This command defines a font for a CJK family which can be activated for typesetting by the command `\CJKfamily{familyname}`.

For a full description on the parameters `` and `font name`, we refer to the package `fontspec`.

The next example shows the effect of some of these commands. For the default English typeface *TeX Gyre Termes* is chosen, the default Chinese typeface is *Bitstream CyberCJK*, while (Song typeface), while *AR PL SungtiL GB* is established as the CJK family “song”.

This is default font abCD.
This is the bold font abCD.
This is the italic font abCD.
And the bold italic font abCD.
 Finally this is Song typeface.

```
\usepackage{xeCJK}
\setmainfont{TeX Gyre Termes}
\setCJKmainfont{Bitstream CyberCJK}
\setCJKfamilyfont{song}{AR PL SungtiL GB}
This is default font abCD. \\
{\bfseries This is the bold font abCD.} \\
{\itshape This is the italic font abCD.} \\
{\bfseries\itshape And the bold italic font abCD.} \\
\CJKfamily{song} Finally this is Song typeface.}
```

Exa.
3-4-1

`xeCJK` offers improved Chinese and English spacing processing, and may avoid the single Chinese character monopolizing a section of last line. The following example shows the effect of the `CJKchecksingle` option.

```
\usepackage[boldfont,slantfont,CJKaddspaces,CJKchecksingle]{xeCJK}
\setCJKmainfont{Bitstream CyberCJK}
\providecommand\mytext{xeCJK 改进了中英文间距的处理，并可以避免单个汉字独占一段的最后一行。}

\section*{First with the option ``checksingle"}
\mytext\par\mytext\par\mytext

\section*{And now without the option ``checksingle"}

\makeatletter
\let\xeCJK@checksingle\xeCJK@notchecksingle
\makeatother
\mytext\par\mytext\par\mytext
```

Exa.
3-4-2

First with the option “checksingle”

xeCJK 改进了中英文间距的处理，并可以避免单个汉字独占一段的最后一行。

xeCJK 改进了中英文间距的处理，并可以避免单个汉字独占一段的最后一行。

xeCJK 改进了中英文间距的处理，并可以避免单个汉字独占一段的最后一行。

And now without the option “checksingle”

xeCJK 改进了中英文间距的处理，并可以避免单个汉字独占一段的最后一行。

xeCJK 改进了中英文间距的处理，并可以避免单个汉字独占一段的最后一行。

xeCJK 改进了中英文间距的处理，并可以避免单个汉字独占一段的最后一行。

3.4.1.2 Advanced settings

```
\punctstyle{PunctStyle}
```

Sets the CJK punctuation style. xeCJK predefines the following *PunctStyle* styles for typesetting punctuation.

quanjiao or fullwidth

typeset all punctuation in full-width, or two adjoint punctuation, the first is typeset in half-width;

banjiao or halfwidth

typeset all punctuation in half-width;

kaiming or mixedwidth

typeset all punctuation in half-width except the period, question, and exclamation marks;

hangmobanjiao or marginkerning

typeset punctuation at the end of lines in half-width.

CCT Use the CCT Chinese T_EX system format (<http://freshmeat.net/projects/ceceetee/>).

plain leave the punctuation untouched *as-is*.

```
\xeCJKallowbreakbetweenpuncts    \xeCJKnobreakbetweenpuncts
```

By default, xeCJK prohibits line breaks between punctuation. The command `\xeCJKallowbreakbetweenpuncts` allows line breaks, while `\xeCJKnobreakbetweenpuncts` disallows them.

```
\xeCJKsetslantfactor{slant factor}
\xeCJKsetemboldenfactor{embolden factor}
```

Sets the slant (a value between -0.999 and 0.999) and embolden factors, respectively. Default settings are

```
\xeCJKsetslantfactor{0.17}
\xeCJKsetemboldenfactor{4}
```

Note that both macros effect only CJK families that are defined subsequently in the \LaTeX source file.

```
\CJKnormalspaces    \CJKaddspaces
```

By default, `xeCJK` leaves spaces between CJK and other characters untouched whereas it ignores spaces between CJK characters. One can use `\CJKaddspaces` to add a space between CJK and other characters if a blank space is not present and use `\CJKnormalspaces` to change back to the default.

```
\CJKsetecglue{value}
```

Allows you to control the spacing between Chinese and English. The default is `\CJKsetecglue`

```
\usepackage[boldfont,slantfont,CJKaddspaces]{xeCJK}
\setCJKmainfont{Bitstream CyberCJK}

\providecommand\mytext{%
这是 English 中文 {\itshape Chinese} 中文 \LaTeX\
间隔 \emph{Italic} 中文\textbf{字体} a 数学 $b$ $c$ $d$
\newline
这是English中文{\itshape Chinese}中文\LaTeX\
间隔\emph{Italic}中文\textbf{字体}a数学$b$ $c$ $d$\newline
This is an example. 这是一个例子
}
```

```
\CJKaddspaces
\CJKsetecglue{\hskip 0.15em plus 0.05em minus 0.05em}
\mytext
```

```
\CJKaddspaces
\CJKsetecglue{ }
\mytext
```

```
\CJKnormalspaces
\mytext
```

```
这是 English 中文 Chinese 中文  $\LaTeX$  间隔 Italic 中文 字体 a 数学 b c d
这是 English 中文 Chinese 中文  $\LaTeX$  间隔 Italic 中文 字体 a 数学 b c d
This is an example. 这是一个例子
这是 English 中文 Chinese 中文  $\LaTeX$  间隔 Italic 中文 字体 a 数学 b c d
这是 English 中文 Chinese 中文  $\LaTeX$  间隔 Italic 中文 字体 a 数学 b c d
This is an example. 这是一个例子
这是 English 中文 Chinese 中文  $\LaTeX$  间隔 Italic 中文 字体 a 数学 b c d
这是English中文 Chinese中文 $\LaTeX$  间隔 Italic中文 字体a数学b c d
This is an example. 这是一个例子
```

Exa.
3-4-3

THE TEXT BELOW WAS TRANSLATED BY BABELFISH FROM THE CHINESE COMPUSCRIPT
ONCE I UNDERSTAND ITS MEANING THE TEXT WILL BE REWRITTEN

One can see that

- `{<text>}` `{<text>}` as well as English `{<text>}` the middle blank space can retain (cannot adjust), but it does not have the blank space, (see above then can according to need to increase surface example).
- in the Chinese and the line the mathematical expression gap control is through defines `\everymath` and `\everydisplay` realization, sometimes is possible invalid, The solution is the manual Canadian blank space.

```
\xeCJKsetcharclass{first}{last}{class}
```

under default state, `\xeCJK 0x2000` — Between the `0xFFFF` character regards as the CJK writing, namely the CJK correlation typeface establishment () to is only effective in this scope character. May use the above great order change character category. For example, the following orders to establish `0x0080` — Between the `0x2FFF` character is the non-CJK writing, but `0x20000` — Between `0x30000` is the CJK writing:

```
\xeCJKsetcharclass {"80"} {"2FFF"} {0}
\xeCJKsetcharclass {"20000"} {"30000"} {1}
```

attention: Last the parameter only can be 0 or 1. Do not change the character category easily.

```
\xeCJKcaption[<encoding>]{caption}
```

is similar with `\CJKcaption`, may choose the parameter to use to choose the code, default is UTF-8.

```
\xeCJKsetkern{punctuation 1}{punctuation 2}{kern}
```

if is unsatisfied to the default disposition, may use this order to establish between two punctuations the distances. For example,

```
\xeCJKsetkern{:}{"}{0.3em}
```

3.4.1.3 Compatibility

CJKfntef

Loads the `CJKfntef` (from the `CJK` package) after `\xeCJK` to get various effects on CJK characters. This package provides the commands `\CJKKunderline` to draw a line under CJK characters, and `\CJKKunderdot` draw a dot below such characters. The effect of these two commands can be combined, as the following example shows.

```
\usepackage[boldfont,slantfont]{xeCJK}
\usepackage{CJKfntef}
\setCJKmainfont{Bitstream CyberCJK}
\setCJKmonofont{Bitstream CyberCJK}
汉字Chinese数学$x=y$空格
```

汉字 Chinese 数学 \$x=y\$ 空格

`\CJKUnderline{汉字Chinese数学$x=y$}加下划线，可以\CJKUnderdot{同时加点}。}`

`\CJKUnderline{汉字 Chinese 数学 $x=y$}加下划线，可以\CJKUnderdot{同时加点}。}`

`\CJKUnderline*{汉字}加下划线，可以\CJKUnderdot{同时加点}。}`

`\CJKUnderdot{汉字}加点，可以\CJKUnderline{同时加下划线}。}`

汉字 Chinese 数学 $x = y$ 空格
 汉字 Chinese 数学 $x = y$ 空格
 汉字 Chinese 数学 $x = y$ 加下划线，可以同时加点。
 汉字 Chinese 数学 $x = y$ 加下划线，可以同时加点。
 汉字加下划线，可以同时加点。
 汉字加点，可以同时加下划线。

Exa.
3-4-4

CJKnumber

To use the package `CJKnumb`, one can specify the option `CJKnumber` while loading `xeCJK`.

12345 12345 一万二千三百四十五。
 67890 67890 六万七千八百九十。

```
\usepackage[CJKnumber]{xeCJK}
\setmainfont{TeX Gyre Termes}
\setCJKmainfont{Bitstream CyberCJK}
12345 $12345$ \CJKnumber{12345}.

67890 $67890$ \CJKnumber{67890}.
```

Exa.
3-4-5

CJK

To be compatible with the CJK-related packages `CJKnumb`, `CJKfntef` and `CJKulem`, `xeCJK` reimplements some macros defined in the package `CJK`. Therefore packages `xeCJK` and `CJK` are incompatible and `xeCJK` will prevent the user from loading `CJK` subsequently.

3.4.2 The `zhspacing` package

A more detailed and expert handling of Chinese typographic peculiarities is possible with Dian Yin's `zhspacing` package (available from <http://code.google.com/p/zhspacing/>), which takes advantage of the \XeTeX command `\XeTeXinterchartoks`.

这是中文测试。中文和English的混排。中文
 和 $E = mc^2$ 的混排。
 这是中文测试。中文和English的混排。中文
 和 $E = mc^2$ 的混排。
 这是中文测试。中文和English的混排。中文和
 $E = mc^2$ 的混排。

```
\usepackage[no-math]{fontspec}
\setmainfont[BoldFont=SimHei]{SimSun}
\usepackage{zhspacing}
\raggedright\noindent
这是中文测试。中文和English的混排。
中文和 $E = mc^2$ 的混排。
\par\noindent
这是中文测试。中文和English的混排。
中文和 $E = mc^2$ 的混排。
\par\zhspacing\noindent
这是中文测试。中文和English的混排。
中文和 $E = mc^2$ 的混排。
```

Exa.
3-4-6

zhspacing can be used in both plain $X_{\text{T}}\text{E}_{\text{X}}$ or $X_{\text{T}}\text{E}_{\text{X}}$. In the latter case the source would look like

```
\input zhspacing.sty
\zhspacing
    input text
\bye
```

This example shows that spaces after Chinese characters are always ignored. Moreover, a noticeable skip is inserted between Chinese characters and English characters as well as math formulas. In fact, all of the following inputs can produce mixed language output with skip automatically inserted between Chinese and English characters.

Exa.
3-4-7

<pre>中Eng文, 中Eng文, 中Eng 文, 中Eng 文 中 Eng 文, 中 Eng 文, 中 Eng 文, 中 Eng 文</pre>	<pre>\usepackage{zhspacing}\zhspacing \begin{flushleft} \emptyskipsscheme 中Eng文, 中 Eng文,\ 中Eng 文, 中 Eng 文\ \simssunskipsscheme 中Eng文, 中 Eng文,\ 中Eng 文, 中 Eng 文 \end{flushleft}</pre>
--	--

Look close at the inputs on the first line and you will see that they generate exactly the same output, as do the inputs on the second line. This means that spaces following Chinese characters are ignored if no spacing scheme is activated (`\emptyskipsscheme`). However, after activation of the spacing scheme (`\simssunskipsscheme`) defined in the `zhspacing` package a skip is introduced for such a space. Note that the skip between `Eng` and `文` on the last two lines is somewhat wider than the skip between `中` and `Eng`. That is because the space is produced by the space token after the letter `g`, not the skip automatically inserted by `zhspacing`'s skip mechanism.

3.4.2.1 Punctuation skip adjustment

Proper Chinese typesetting requires consecutive fullwidth punctuations be compressed, and a linebreak before or after a fullwidth punctuation will cut off the blank spaces of this punctuation, making it align to the margin. `zhspacing` solved these problems, as well as proper prohibitions(禁则). Here's an example.

Exa.
3-4-8

他强调，“三个代表”重要思想是在新的历史条件下运用马克思主义的立场、观点和方法的典范，是我们学习马克思主义的立场、观点和方法最现实、最生动的教材。“三个代表”重要思想是与时俱进的理论。

3.4.2.2 Advanced usage

Fonts

`zhspacing` uses an extensible way of selecting fonts. The rules can be summarized as follows,

- Western characters, i.e., those that are not CJKV ideograms nor CJKV punctuation use the default font.
- Chinese characters use separate fonts. Font changes in the document does not affect the font used to display Chinese, unless you are using the NFSS scheme to change font series or shape.
- When typesetting basic Chinese ideograms the command `\zhfont` is executed.

- When typesetting Chinese punctuations the command `\zhpunctfont` is executed.
- When typesetting CJK Ext-A characters the command `\zhcjkextafont` is executed.
- When typesetting CJK Ext-B characters the command `\zhcjkextbfont` is executed.
- When switching from non-Chinese to Chinese characters the command `\zhs@savefont` is executed, whereas when switching back the command `\zhs@restorefont` is executed.

`zhspacing`'s default definitions in $\text{Xe}_{\text{L}}\text{TeX}$ for these commands are:

```
\newfontfamily\zhfont[BoldFont=SimHei]{SimSun}
\newfontfamily\zhpunctfont{SimSun}
\def\zhcjkextafont{\message{CJK Ext-A}}
\def\zhcjkextbfont{\message{CJK Ext-B}}
\def\zhs@savefont{\zhs@savefont{old}}
\def\zhs@restorefont{\zhs@restorefont{old}}
```

The internal macros `\zhs@savefont` and `\zhs@restorefont` save and restore the NFSS-related information for the current font.

The extension CJK Ext-A/B fonts are not defined by default since not every user has necessarily installed the fonts needed. The package author recommends to use `Sun-ExtA` and `Sun-ExtB` for these fonts. You can define the ext-font macros manually in a similar way to the definition of `\zhfont`.

Skips

The `zhspacing` package uses a flexible skip mechanism which is based on a series of commands rather than on skip registers. This allows the skips to vary according to the current font size. The list of available skip commands follows. They are all defined according to the following model `\def\skipxxx{\hskip xxxxx}`.

- `\skipzh` Skip between adjacent Chinese characters.
- `\skippenzh` Skip between a Chinese character and a Western character or a math formula.
- `\skipzhopen` Skip before fullwidth opening punctuations, such as ””, ” (”, ” 《”, etc.
- `\skipzhinteropen` Skip before a fullwidth opening punctuation when preceded by another fullwidth punctuation.
- `\skipzhlinestartopen` Skip before a fullwidth opening punctuation when it occurs at the start of a line.
- `\skipzhclose` Skip after fullwidth closing punctuations, such as ””, ”)”, ” 》”, etc.
- `\skipzhinterclose` Skip after a fullwidth closing punctuation when followed by another fullwidth punctuation.
- `\skipzhlineendclose` Skip after a fullwidth closing punctuation when it occurs at the end of a line.
- `\skipzhjudou` Skip after fullwidth judou(句读) punctuations, such as ”、”, ” ,”, ”。”, etc.
- `\skipzhinterjudou` Skip after a fullwidth judou punctuation when followed by another fullwidth punctuation.
- `\skipzhlineendjudou` Skip after a fullwidth judou punctuation when it occurs at the end of a line.
- `\skipnegzhlinestartopen` Negative skip to `\skipzhlinestartopen`.
- `\skipnegzhlineendclose` Negative skip to `\skipzhlineendclose`.

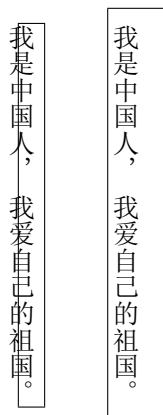
`\skipnegzhlineendjudou` Negative skip to `\skipzhlineendjudou`.

The `zhspacing` package comes with three pre-defined skip schemes, namely `\simsunskipscheme`, `\emptyskipscheme` and `\haltskipscheme`. The first scheme should be suitable for font `SimSun` and other popular Chinese fonts used in China, which does not support OpenType features of `halt`, and needs negative spaces be inserted before opening punctuations and after closing or `judou` punctuations. The second scheme simply adds zero length. And the last one should be fit for OpenType Chinese fonts supporting the `halt` feature such as *Adobe Song Std*, where positive spaces should be inserted before or after certain punctuations. You can define your own skip schemes for customization, of course.

Vertical Chinese

Vertical Chinese can be achieved by adding the raw feature `vertical` for the specified Chinese font. An example is the flowing, which also shows what T_EX thinks the boundingbox of the characters is.

Exa.
3-4-9



```

\usepackage[dvipdfm]{graphicx}
\usepackage{zhspacing}\zhspacing
\newfontfamily\zhfont[RawFeature={vertical:}]{SimSun}
\newfontfamily\zhpunctfont[RawFeature={vertical:
+vert:+vhal}]{Adobe Song Std}

\haltskipscheme
\setlength\fbboxsep{0mm}
\fbbox{\rotatebox{-90}{我是中国人，我爱自己的祖国。}}%
\quad
\setlength\fbboxsep{2mm}
\fbbox{\rotatebox{-90}{我是中国人，我爱自己的祖国。}}

```

Note that in this example, in order to have proper vertical punctuations, we set `\zhpunctfont` to use the *Adobe Song Std* font, which supports the `vert` feature, and change the skip scheme to `\haltskipscheme` to match the `vhal` feature specified. Some Chinese fonts have bugs for typesetting vertical Chinese containing punctuations. Moreover, often the baseline of vertical Chinese is not correct, so that mixing Chinese and English in vertical mode can generate ugly results, and thus should be avoided.

Some more vertical typesetting is shown in the following example, which also explains how easy it is to make X_YT_EX print HTML character references, a possibility that comes in handy if you want to typeset some text from a Web page, where non-Latin characters are sourced using this kind of representation of Unicode characters, which is extremely portable (only ASCII characters are in the HTML

source), and is thus quite often used.

This is English. これは日本語です。

This is English. これは日本語です。

```
\usepackage[dvipdfm]{graphicx}
\usepackage{fontspec}

\fontspec[Mapping=tex-text,Script=CJK]{Kozuka Mincho Pro-VI}

% macro hacking to read chars represented as character references
\catcode\&=\active % make & active
\catcode\#=12 % make # "other"
\def&#\char % replace sequence &# by \char
\catcode\;=\active % make ; active
\def;\relax % and make it a no-operation

\fbxsep0pt
\fbx{This is English.
&#12371;&#12428;&#12399;&#26085;&#26412;&#35486;&#12391;&#12377;&#12290;}

\fontspec[Mapping=tex-text,Vertical=RotatedGlyphs,Script=CJK]{Kozuka
Mincho Pro-VI}

\rotatebox{-90}{\fbx{This is English.
&#12371;&#12428;&#12399;&#26085;&#26412;&#35486;&#12391;&#12377;&#12290;}}
```

Exa.
3-4-10

3.4.2.3 Compatibility

Theoretically, `zhspacing` should be compatible with all macro packages, except those who change the definition of `\hskip` and `\penalty`, in which case special treatment should be applied. I haven't found any conflict when using common packages such as `hyperref` and `fancyhdr`. However, `ulem` redefines `\hskip` and `\penalty`, and causes unexpected output. Use `zhulem` provided along with `zhspacing` instead.

Using `zhspacing` with the `ctex` package needs some precautions, see the manual for more details (<http://www.ctex.org>).

3.4.2.4 Character classes and class inheritance

The actual situation concerning Chinese typesetting is so complicated that it's difficult to figure out exactly how many classes are needed and what we should do when changing from this class to that. In fact, in a more natural way, we can consider from the top down — first there are fullwidth and halfwidth characters as well as boundaries — and construct a hierarchical forest where each node represents a character class. In this way common behaviors can be performed between different families of classes, and specific action can be taken for a particular class pair. That is the idea of *class inheritance*, the concept behind `zhspacing`.

3.5 Examples of the use of Unicode

3.5.1 Unicode fonts and editors

- `emacs` and `vi` when adequate fonts are installed on the system (and made known to the applications)
- `yudit`, a freeware editor (<http://yudit.org>) for Linux and Microsoft Windows

- Resources for Unicode fonts
 - Bitstream *Cyberbit*¹
 - a more recent version of the above *TITUS Cyberbit Basic* (developed at the University of Frankfurt, Germany, see the URL <http://titus.uni-frankfurt.de/>)
 - the shareware fonts *Code2000* for Unicode plane 0, *Code2001* for plane 1, and *Code2002* for plane 2 (see <http://www.code2000.net/>)
 - *Arial Unicode MS*, which comes with the Microsoft's *Windows XP* and *Vista* systems
 - Web page *WAZU JAPAN's Gallery of Unicode Fonts* (<http://www.wazu.jp/index.html>)
 - Web page of Luc Devroye (<http://www.cccg.ca/~luc/fonts.html>)
 - Web page of Alan Wood (<http://www.alanwood.net/unicode/fonts.html>)
 - Web page *Unicode tools and fonts* (<http://www.unifont.org/>)

3.5.2 Examples of Unicode texts

- The Office of the High Commissioner for Human Rights in Geneva publishes the *Universal declaration of human rights* (<http://www.ohchr.org/french>). The site of the Unicode Consortium makes the *Universal declaration of human rights* available in 324 languages to show the power of Unicode (<http://www.unicode.org/udhr>).
- The site www.sacred-texts.com contains hundreds of sacred texts, many in UTF-8. There is Homer in ancient Greek ([cla/homer/greek](http://www.sacred-texts.com/cla/homer/greek)), a multi-language bible in English, French, Hebrew, and Latin ([bib/poly](http://www.sacred-texts.com/bib/poly)), the Coran in Arabic and English ([isl/uq](http://www.sacred-texts.com/isl/uq)), Confucius in Chinese and English ([cfu/cfu.htm](http://www.sacred-texts.com/cfu/cfu.htm)), the Rig Veda in Sanskrit ([hin/rvsan](http://www.sacred-texts.com/hin/rvsan)), etc.
- The Titus project of Indo-Germanic studies (titus.uni-frankfurt.de) and the Perseus Project (http://www.perseus.tufts.edu/cache/perscoll_Greco-Roman.html) contain many classical texts.

¹See <ftp://ftp.netscape.com/pub/communicator/extras/fonts/windows/cyberbit.zip>

Unicode mathematics

4.1 Unicode for handling math across platforms and applications	91
4.2 X _Y TeX handling mathematics fonts	92

4.1 Unicode for handling math across platforms and applications

- It is important to represent math correctly on the Web and in the various typesetting applications.
- T_EX exists for books and MathML (presentation and context) for XML-enabled applications.
- Murray Sargent (Microsoft), member of the W3C MathML Working Group, and his collaborators have developed an extension for OpenType fonts to enable them to handle math (their approach is based on T_EX's math typesetting algorithm as described in Appendix G of the T_EXBook [4]). An additional OpenType MATH table contains the parameters needed to typeset math. This effort resulted in the *Cambria Math* math font.
- Barbara Beeton, Asmus Freytag, and Murray Sargent wrote a paper *Unicode support for mathematics* (www.unicode.org/reports/tr25/tr25-7.html) which describes the default math properties for Unicode characters.
- Murray Sargent describes in the Unicode report *Unicode Nearly Plain-Text Encoding of Mathematics* (unicode.org/notes/tn28/) how with a few additions to Unicode mathematical expressions can usually be represented with a readable Unicode nearly plain-text (linear) format.
- Office 2007 now has a built-in math-engine (see Marray's presentation *Math Editing and display in Office 2007* (research.microsoft.com/workshops/fs2006/presentations/17_Sargent_071706.ppt) and his blog (<http://blogs.msdn.com/murrays>). This *ad hoc* processor is based on T_EXBook's Appendix G algorithm and uses the *Cambria Math* math font and uses the software component MathFont.dll to communicate between the various applications programs.
- The Microsoft Word2007 work and the definition of the OpenType MATH table are unpublished. Paul Topping, in the interest of the scientific community at large wrote a position paper *Design Science Proposal to Microsoft to Help STEM (Scientific/Technical/ Engineering/Mathematical) Publishers Work with Office 2007 Documents* where he asks Microsoft to share the information in its specifications.¹

¹See http://www.dessci.com/en/reference/white_papers/STMOffice2007Proposal.htm

4.2 X_YTeX handling mathematics fonts

- X_YTeX uses the algorithm in Appendix G of the TeXBook to typeset mathematics;
 - for a “standard” TeX math font X_YTeX use the metric information for each character in the corresponding `.tfm` file, then `xdvipdfmx` refers to the `.pfb` file via the virtual font files (when necessary) and the file `dvipdfm.map`,
 - for an OpenType math font, such as *Cambria Math*, X_YTeX reads the metric parameters in the `MATH` table and transforms them into the values needed by the Appendix G algorithm.
- Currently X_YTeX does not use a specific processor to handle OpenType math fonts, but perhaps such support will later be included in the system middleware (ICU).
- Other Unicode math fonts:
 - the font developed by STIX (*Scientific and Technical Information Exchange font*, see <http://www.stixfonts.org/>). This is project where several scientific publishers have co-financed a Unicode-based mathun font that contains over 8000 different glyphs
 - Apostoulos Syropoulos (asyropolous@yahoo.com) is developing another font (*Asana-Math*) with the help of `fontforge` that includes the OpenType `MATH` tables.
- Will Robertson is working on a \LaTeX package `unicode-math` to provide a simple interface to OpenType math fonts with \LaTeX .

Bibliography

- [1] Adobe Systems. *Adobe Type 1 Font Format*. Addison-Wesley, Reading, MA, 1990.
This so-called “White Book” contains the specification of Adobe’s Type 1 font format, including information about hints, the encryption mechanism, encodings, and the flex procedure. Available electronically from
http://partners.adobe.com/public/developer/en/font/T1_SPEC.PDF
- [2] Youssef Jabri. “The Arabi system. T_EX writes in Arabic and Farsi”. *TUGboat*, 27(4):147–153, 2006.
This article describes the *Arabi* package, which introduces support in the L^AT_EX system for languages using the Arabic script, in particular Arabic and Farsi. The package comes with a set of good-quality free fonts, but may also use commercial fonts. It supports many 8-bit input encodings, e.g., CP-1256, ISO-8859-6 and Unicode UTF-8, and can typeset classical Arabic poetry.
<http://www.tug.org/TUGboat/Articles/tb27-2/tb87jabri.pdf>
- [3] Gabriel Mandel Khan. *Arabic Script*. Abbeville Press Publishers, New York, 2001.
This book provides a detailed look at the Arabic script and its calligraphy, an essential part of the Arabic culture. Since Arabic is the language of the Koran, with the spread of Islam to large parts of the world, the Arabic script is now one of the world’s major forms of writing. With the help of over 300 two-color and black-and-white pictures the author describes each letter, its history, meaning, variants, and calligraphic adaptations, as well as its philosophical, theological, and cultural significance.
The book starts with a short introduction sketching the development of the Arabic alphabet and the various scripts in which it has been written. Then, the first major part of the book, “The Letters of the Alphabet”, is devoted to the treatment of individual letters and their shapes which can vary depending on the letter’s position within a word. Over thirty different styles, or scripts, are illustrated for each letter. The letter’s pronunciation, its characteristic in reciting the Koran, plus possible other cultural associations are defined. The second major part of the book, “Styles, variants, and calligraphic adaptations”, provides an large set of historic examples of Arabic writing. Finally, there is a glossary and an index.
- [4] Donald E. Knuth. *The T_EXbook, volume A of Computers and Typesetting*. Addison-Wesley, Reading, MA, 1986.
This book is the definitive user’s guide and complete reference manual for T_EX.
- [5] Frank Mittelbach, Michel Goossens, Johannes Braams, David Carlisle, and Chris Rowley. *The L^AT_EX Companion, Second Edition*. Addison-Wesley, Reading, MA, 2004.
This book describes over 200 L^AT_EX packages and presents a whole series of tips and tricks for using L^AT_EX in both traditional and modern typesetting, in particular how to customize layout features to your own needs—from phrases and paragraphs to headings, lists, and pages. It provides expert advice on using L^AT_EX’s basic formatting tools to create all types of publication, from memos to encyclopedias. It covers in depth important extension packages for tabular and technical typesetting, floats and captions, multi-column layouts, including reference guides and discussion of the underlying typographic concepts. It details techniques for generating and typesetting indexes, glossaries, and bibliographies, with their associated citations.
- [6] Peter D. Daniels and William Bright. *The World’s Writing Systems*. Oxford University Press, New York, 1996.
A detailed description of the major historical and modern writing systems of the world. The more than eighty articles contributed by expert scholars in the field are organized in twelve units, each dealing with a particular group of writing systems defined historically,

geographically, or conceptually. Each unit begins with an introductory article providing the social and cultural context in which the group of writing systems was created and developed. Articles on individual scripts detail the historical origin of the writing system in question, its structure (with tables showing the forms of the written symbols), and its relationship to the phonology of the corresponding spoken language. Each writing system is illustrated by a passage of text, accompanied by a romanized version, a phonetic transcription, and a modern English translation. Each article concludes with a bibliography.

Units are arranged according to the chronological development of writing systems and their historical relationship within geographical areas. First, there is a discussion of the earliest scripts of the ancient Near East. Subsequent units focus on the scripts of East Asia, the writing systems of Europe, Asia, and Africa that have descended from ancient West Semitic ("Phoenician"), and the scripts of South and Southeast Asia. Other units deal with the recent and ongoing process of decipherment of ancient writing systems; the adaptation of traditional scripts to new languages; new scripts invented in modern times; and graphic systems for numerical, music, and movement notation.

- [7] The Unicode Consortium. *The Unicode Standard, Version 5.0*. Addison-Wesley, Reading, MA, 2007.

The reference guide of the Unicode Standard, a universal character-encoding scheme that defines a consistent way of encoding multilingual text. Unicode is the default encoding of HTML and XML. The book explains the principles of operation and contains images of the glyphs for all characters presently defined in Unicode.

Available for restricted use from: <http://www.unicode.org/versions/Unicode5.0.0/>

Index of Commands and Concepts

The index has been split into two parts. We start with a general index that covers all entries. We end with an index of authors.

To make the indexes easier to use, the entries are distinguished by their “type”, and this is often indicated by one of the following “type words” at the beginning of the main entry or a sub-entry:

boolean, counter, document class, env., file, file extension, font, key, key value, option, package, program, rigid length, or syntax.

The absence of an explicit “type word” means that the “type” is either a \TeX “command” or simply a “concept”.

Use by, or in connection with, a particular package is indicated by adding the package name (in parentheses) to an entry or sub-entry. There is one “virtual” package name, `tlgc`, that indicates commands introduced only for illustrative purposes in this book.

A *blue italic* page number indicates that the command or concept is demonstrated in an example on that page.

When there are several page numbers listed, **bold** face indicates a page containing important information about an entry, such as a definition or basic usage.

When looking for the position of an entry in the index, you need to realize that, when they come at the start of a command or file extension, both the characters `\` and `.` are ignored. All symbols come before all letters and everything that starts with the `@` character will appear immediately before `A`.

Symbols

.fonts.conf file, 23
 .log file, 44
 \langle , 63
 $\$HOME/.fonts.conf$ file, 23

A

$\backslash active$, 26
 $\backslash aemph$, 65, 66
 Aleph program, 47
 amsart document class, 55
 amsbook document class, 55
 amsmath package, 55, 56
 amsthm package, 55
 $\backslash arab$, 66
 arab env., 64, 65, 66, 68, 69, 70, 71
 Arabi package, 93
 arabi package, 62
 $\backslash arabicfont$, 64, 65, 66, 68, 69, 71, 73
 arabtex package, 62–64, 65, 68, 75
 arabtext env., 63
 arabxetex package, ii, xi, 64–78
 arabxetex.sty package, 64
 array package, 55, 71
 article document class, 55
 ATSUI program, 21, 30, 31, 34, 37
 $\backslash autofootnoterule$, 56, 59

B

babel package, 43
 beamer document class, 55
 beamerbaseauxtemplates package, 55
 beamerbasemplates package, 55
 beamerthemebidiJLTree package, 55
 beamerthemeJLTree package, 55
 $\backslash beginL$, 22
 $\backslash beginR$, 22
 $\backslash bfseries$, 80
 bidi package, ii, 55–61, 64
 bidi2in1 package, 55
 bidibeamer document class, 55
 bidibeamerbaseauxtemplates package, 55
 bidibeamerbasemplates package, 55
 bidimemoir document class, 55
 bidimoderncv document class, 55
 bidipresentation package, 55
 book document class, 55
 bookest document class, 55
 booktabs package, 55
 $\backslash bye$, 85

C

$\backslash catcode$, 26
 $\backslash char$, 25
 $\backslash chardef$, 25
 CJK package, 83, 84
 $\backslash CJKaddspaces$, 82, 83
 CJKchecksingle option, 80
 $\backslash CJKfamily$, 80
 CJKfntef package, 83, 84
 $\backslash CJKnormalspaces$, 82, 83

CJKnumb package, 79, 84
 $\backslash CJKnumber$, 84
 $\backslash CJKsetecglue$, 82, 83
 CJKulem package, 84
 $\backslash CJKunderdot$, 83
 $\backslash CJKunderline$, 83
 $\backslash cline$, 60
 color package, 58
 crop package, 22
 ctex package, 88
 cvthemebidicasual package, 55
 cvthemebidiclassic package, 55
 cvthemecasual package, 55
 cvthemeclassic package, 55
 cyr-lat-iso9 file, 29
 cyr-lat-iso9.tex file, 29

D

dcolumn package, 55
 $\backslash defaultfontfeatures$, 44
 draftwatermark package, 55
 dvipdfm.map file, 92
 dvipdfmx program, 21
 dvips program, 1, 27

E

emacs program, 88
 $\backslash emptyskipsscheme$, 85, 87
 $\backslash endL$, 22
 $\backslash endR$, 22
 euenc package, 43
 $\backslash everydisplay$, 83
 $\backslash everymath$, 83
 extbook document class, 55

F

fancyhdr package, 55, 88
 $\backslash farsi$, 64
 farsi env., 64, 73
 $\backslash farsifont$, 73
 $\backslash fbox$, 68, 88
 fc-cache program, 24
 fc-list program, 24
 fc-match program, 24
 .fd file extension, 22
 flushleft env., 68
 fmultico package, 57
 $\backslash font$, 21, 27, 29, 34–36, 38, 79
 fontconfig program, 23, 24, 27, 31
 fontforge program, 92
 fontinst package, 43
 fontinst program, 21
 fonts.conf file, 23, 24
 $\backslash fontspec$, 44, 45, 46
 fontspec package, ii, xi, 22, 33, 43–46, 64, 66–68, 71, 80, 84, 88
 fontspec.cfg file, 43, 44
 FontTools program, 16
 $\backslash footnote$, 59
 freetype program, 23, 32
 frhyph.tex file, 26

`\fullvocalize`, 63, 65

G

geometry package, 22
graphics package, 22
graphicx package, 55

H

`\haltskipsscheme`, 87
`\hamzaB`, 69
hhline package, 55
`\hline`, 60
`\hskip`, 88
hyperref package, 22, 88

I

ICU program, 21, 26, 29–31, 34, 37, 46, 92
`\ifthenelse`, 38
`\input`, 85
`\itshape`, 80

K

`\kashmiri`, 65
kashmiri env., 65
kpathsea program, 33
`\kurdish`, 65
kurdish env., 65

L

`\lccode`, 26
`\leftfootnoterule`, 59
listings package, 55
localfonts.conf file, 23, 24
longtable package, 55
`\LR`, 57
`\LRE`, 57
LTR env., 57
`\LTRdblcol`, 57
`\LTRfootnote`, 59

M

`\malay`, 65
malay env., 65
`\mathalpha`, 39
`\mathbin`, 39
`\mathclose`, 39
MathFont.dll program, 91
`\mathop`, 39
`\mathopen`, 39
`\mathord`, 39
mathpazo package, 43
`\mathpunct`, 39
`\mathrel`, 39
memoir document class, 55
metalogo package, 43
minitoc package, 55
moderncv document class, 55
multicol package, 57
multicols env., 57
`\multicolumn`, 60, 71
multirow package, 55

`myfont.ttx` file, 17
`myfontmods.ttx` file, 17

N

`\newfontface`, 45, 46
`\newfontfamily`, 45, 64, 65, 66, 68, 69, 71, 73, 86, 87
`\novocalize`, 65

O

Office program, 12, 13
Omega program, 47
OpenOffice program, 14
Openoffice program, 13
OpenType-info.tex file, 39
otfinfo program, 13
`\ottoman`, 65
ottoman env., 65

P

packages

amsmath, 55, 56
amsthm, 55
Arabi, 93
arabi, 62
arabtex, 62–64, 65, 68, 75
arabxetex, ii, xi, 64–78
arabxetex.sty, 64
array, 55, 71
babel, 43
beamerbaseauxtemplates, 55
beamerbasemplates, 55
beamerthemebidiJLTree, 55
beamerthemeJLTree, 55
bidi, ii, 55–61, 64
bidi2in1, 55
bidibeamerbaseauxtemplates, 55
bidibeamerbasemplates, 55
bidipresentation, 55
booktabs, 55
CJK, 83, 84
CJKfntef, 83, 84
CJKnumb, 79, 84
CJKulem, 84
color, 58
crop, 22
ctex, 88
cvthemebidicasual, 55
cvthemebidiclassic, 55
cvthemecasual, 55
cvthemeclassic, 55
dcolumn, 55
draftwatermark, 55
euenc, 43
fancyhdr, 55, 88
fmultico, 57
fontinst, 43
fontspec, ii, xi, 22, 33, 43–46, 64, 66–68, 71, 80, 84, 88
geometry, 22
graphics, 22
graphicx, 55
hhline, 55
hyperref, 22, 88

packages (*cont.*)

- listings, 55
- longtable, 55
- mathpazo, 43
- metalogo, 43
- minitoc, 55
- multicol, 57
- multirow, 55
- pdfpages, 55
- pgf, 22
- pstricks, 55
- ragged2e, 55
- stabular, 55
- supertabular, 55
- tabls, 55
- tabularx, 55
- tabulary, 55
- threeparttable, 55
- tikz, 55
- tlgc, 95
- tocloft, 55
- tocstyle, 55
- ulem, 88
- unicode-math, xi, 43, 92
- vwcol, 58
- wrapfig, 55
- xcolor, 22, 58
- xeCJK, 79–84
- xecjk, ii
- xecolour, 58
- xecyk, 79
- xltxtra, 43
- xunicode, 43, 56
- zhspacing, ii, xi, 84–88
- zhulem, 88
- \pashto, 65
- pashto env., 65
- \pdflastxpos, 42
- \pdflastypos, 42
- pdflatex program, 2
- \pdfpageheight, 42
- pdfpages package, 55
- \pdfpagewidth, 42
- \pdfsavepos, 42
- \penalty, 88
- .pfb file extension, 92
- pgf package, 22
- \pounds, 28
- pstricks package, 55
- \punctstyle, 81

R

- ragged2e package, 55
- \raisebox, 60
- rapport3 document class, 55
- \rcases, 59
- refrep document class, 55
- report document class, 55
- \rightfootnoterule, 59
- \RL, 57
- rldocument option, 56
- \RLE, 57

- \rmfamily, 45
- \rotatebox, 87, 88
- RTL env., 57
- \RTLdblcol, 56, 57
- RTLdocument option, 56
- \RTLfootnote, 59

S

- \Salam, 67
- scrartcl document class, 55
- scrbook document class, 55
- scrrprt document class, 55
- \setarab, 63
- \setCJKfamilyfont, 80
- \setCJKmainfont, 80, 82–84
- \setCJKmonofont, 80
- \setCJKsansfont, 80
- \setfootnoteLR, 59
- \setfootnoteRL, 59
- \setLR, 56
- \setLTR, 56, 57
- \setmainfont, 22, 44, 45, 66, 68, 71, 80, 84
- \setmonofont, 44, 45, 80
- \setnash, 63
- \setnashbf, 63
- \setRL, 56
- \setRTL, 56–60
- \setsansfont, 44, 45, 80
- \SetTranslitStyle, 65
- \sfcode, 26
- \sffamily, 45
- \simsunskipscheme, 85, 87
- \sindhi, 65
- sindhi env., 65
- \skippenzh, 86
- \skipnegzhlineendclose, 86
- \skipnegzhlineendjudou, 87
- \skipnegzhlinestartopen, 86
- \skipzh, 86
- \skipzhclose, 86
- \skipzhinterclose, 86
- \skipzhinterjudou, 86
- \skipzhinteropen, 86
- \skipzhjudou, 86
- \skipzhlineendclose, 86
- \skipzhlineendjudou, 86, 87
- \skipzhlinestartopen, 86
- \skipzhopen, 86
- stabular package, 55
- supertabular package, 55

T

- tabls package, 55
- tabular env., 60, 71
- tabularx package, 55
- tabulary package, 55
- TECkit program, 28, 64
- tex program, 33
- tex-text-tec file, 28
- tex-text.map file, 28
- tex-text.tec file, 28
- texmf file, 27

`\text`, 59
`\textarab`, 64
`\textroman`, 64, 67
`\textwidth`, 59
`\textwidthfootnoterule`, 59
`\TeXeTstate=1`, 22
`.tfm` file extension, 22, 37, 43, 92
`tfm` file, 27
threeparttable package, 55
tikz package, 55
tlgc package, 95
tocloft package, 55
tocstyle package, 55
`\transtrue`, 63, 65
`.ttc` file extension, 18
ttc2ttf program, 18
ttx program, 16, 17

U

`\UC`, 65
`\uccode`, 26
`\uighur`, 65
uighur env., 65
ulem package, 88
unicode-math package, xi, 43, 92
`\unsetfootnoteRL`, 59
`\unsetLTR`, 56
`\unsetRL`, 56
`\unsetRTL`, 56
`\urdu`, 65
urdu env., 65, 73
`\urdufont`, 73
`\usepackage`, 55

V

`.vf` file extension, 22, 43
vi program, 88
`\vocalize`, 65
vwcol env., 58
vwcol package, 58

W

W32tex program, 21
Web2C program, 33
wrapfig package, 55

X

xcolor package, 22, 58
`.xdv` file extension, 27
xdvipdmtx program, 27, 32, 33, 92
xeCJK package, 79–84
xecjk package, ii
`\xeCJKallowbreakbetweenpuncts`, 81
`\xeCJKcaption`, 83
`\xeCJKnobreakbetweenpuncts`, 81
`\xeCJKsetcharclass`, 83
`\xeCJKsetemboldenfactor`, 82
`\xeCJKsetkern`, 83
`\xeCJKsetslantfactor`, 82

xecolour package, 58
xecyck package, 79
xelatex program, 19–95
`\XeTeX`, 41, 43
XeTeX program, 19, 21
xetex program, 19–95
`\XeTeXcharclass`, 41
`\XeTeXcharglyph`, 37
`\XeTeXdashbreakstate`, 41
`\XeTeXdefaultencoding`, 41
`\XeTeXdelcode`, 39
`\XeTeXdelcodenum`, 39
`\XeTeXdelimitter`, 39
`\XeTeXfonttype`, 37, 38
`\XeTeXglyph`, 36, 37
`\XeTeXglyphindex`, 36, 37
`\XeTeXinputencoding`, 26, 41
`\XeTeXinterchartokenstate`, 41
`\XeTeXinterchartoks`, 41, 84
`\XeTeXlinebreaklocale`, 42
`\XeTeXlinebreakpenalty`, 42
`\XeTeXlinebreakskip`, 30, 42
`\XeTeXmathchardef`, 39
`\XeTeXmathcode`, 39
`\XeTeXmathcodenum`, 39
`\XeTeXOTcountfeatures`, 38
`\XeTeXOTcountlanguages`, 38
`\XeTeXOTcountscritps`, 38
`\XeTeXOTfeaturetag`, 39
`\XeTeXOTlanguagetag`, 38
`\XeTeXOTscripttag`, 38
`\XeTeXpdffile`, 42
`\XeTeXpicfile`, 42
`\XeTeXradical`, 41
`\XeTeXrevision`, 26, 41
`\XeTeXupwardsmode`, 42
`\XeTeXuseglyphmetrics`, 36
`\XeTeXuseglyphmetricsfont`, 36
`\XeTeXversion`, 41
xltextra package, 43
xu-frhyph.tex file, 26
xu-hyphen file, 26
xu-t1.tex file, 26
xunicode package, 43, 56

Y

yudit program, 88

Z

`\zhcjkextafont`, 86
`\zhcjkextbfont`, 86
`\zhfont`, 85, 86, 87
`\zhpunctfont`, 86, 87
`\zhs@restorefont`, 86
`\zhs@restorefont`, 86
`\zhs@savefont`, 86
`\zhs@savefont`, 86
`\zhspacing`, 84, 85, 87
zhspacing package, ii, xi, 84–88
zhulem package, 88

People

Beeton, Barbara, 91
Berry, Karl, 27, 33
Buchbinder, Adam, xi

Charette, François, ii, xi, 64
Cho, Jin-Hwan, 21

Devroye, Luc, 54

Ferres, Leo, ii, xi
Freytag, Asmus, 91

Goossens, Michel, 93

Jabri, Youssef, 62

Kabel, Rik, xi
Kakuto, Akira, 21
Kew, Jonathan, ii, xi, 19, 21, 43, 47
Khalighi, Vafa, ii, 55
Knuth, Donald, 1, 93

Lagally, Klaus, 62, 64

McCreehy, David, 54
Mittelbach, Frank, 93
Moore, Ross, 21, 43

Piška, Karel, ii, xi

Robertson, Will, ii, 22, 43, 92

Sargent, Murray, 91
Shigeru, Miyata, 21
Sun, Wenchang, 79

Topping, Paul, 91

Weiss, Mimi, 54
Wood, Alan, 5

Yin, Dian, ii, xi, 84