Catching up to Unicode[*]

Roozbeh Pournader
Computing Center
Sharif University of Technology
Azadi Avenue
Tehran, Iran
roozbeh@sharif.edu
http://sina.sharif.edu/~roozbeh/

Abstract

Unicode, the universal character set standard first published in 1991, has changed dramatically in its more than ten years of development, trying to achieve maximum interoperability of internationalized text between different platforms. In the meanwhile, TEX, and its companion production tools, have stuck loyally to their roots of special formats and traditions known only to the TEX community, rather ignoring this moving target.

This paper will try to draw a general outline of the Unicode standard in its present situation, emphasizing on its recently introduced features. It will also try to specify new requirements for Omega, in order to make it usable in the developing realm of *standard renderings*.

Introduction

The Unicode Standard (The Unicode Consortium, 2000, as amended by Unicode Editorial Committee, 2001, and Unicode Editorial Committee, 2002), and the big family gathered around it, ranging from MathML (Carlisle et al., 2001) to OpenType (Adobe Systems Incorporated and Microsoft Corporation, 2001), have been successful in making document interchange truly global. Now, you don't need to worry about the very basics of how to encode your text if you want to develop an application handling Lao, or send an email containing APL symbols; you won't need to educate the user base about semantic markup, as even Microsoft is now recommending XML; and you don't even need to develop new tools for your basic Syriac text processing needs, IBM has already developed many general purpose ones if you can't find some suitable ones in your Linux machine, which you only need to configure.

The Unicode standard, while keeping the same encoding model of the beginning days, has become very complicated recently, because of the advanced requirements of text processing applications and the natural languages and scripts themselves. Also, because of the wide deployment of Unicode by Microsoft, a Unicode contributor, starting with Windows 2000 and Office 2000 series of products, many loopholes and implementation difficulties have been discovered, and fixed in the standard.

Unfortunately, Microsoft has got some of the ideas wrong, but fortunately, many of us don't live in a Microsoft world. Linux and GNU communities, with their affinity for standards, and now backed by corporations like IBM and Sun, have helped implement Unicode in a compliant way. With an out of the box installation of Red Hat Linux 7.3, you can now edit a Unicode text document in Ethiopic containing some Hebrew, some Braille, and a few chess symbols using vim under xterm[2]. But sadly, very few tools exist to help you make a typographically beautiful printout of the file.

The current Unicode

Putting history aside, Unicode, in its latest 3.2 version, is a character set assigning a number from 0 to $\texttt{10FFFF}_{16} = 17 \times 2^{16} - 1 = 1,114,111$ to each character[3]. (We will be using the notation U+20A8 to refer to the character coded as $\texttt{20A8}_{16}$, which (as it happens) is a Rupee Sign). The characters are distributed in blocks of related functionality, such as

[*] Preparation of this paper, its prerequisite research, and its presentation at TUG 2002 have been supported by the FarsiWeb Project, Science and Arts Foundation (http://www.farsiweb.info/).

[2] The same is of course possible under Microsoft Windows XP if you have the appropriate fonts and keyboard mapping programs.

[3] At present, the standard intends never to change this upper limit, barring extraterrestial scripts!

a script like Cyrillic or a common usage like Mathematical Operators.

Unicode does (and will) not encode any presentation forms (also known as *glyphs*) unless for backward compatibility with legacy character sets. So, you will not see four different characters for the Syriac letter Beth, but only one.

On the other hand, there are many kinds of special characters encoded, from control characters for specifying the shape of newly-invented CJK ideographs to weather forecast symbols. There are also more than 70,000 characters reserved for private use, which lets two parties interchange text without requiring the encoding of their limited-use characters by the standard.

Unicode also assigns many normative and informative properties to each character, together with descriptions of characters and blocks. For example, the Arabic block contains a very exact description of a minimum Arabic contextual shaping algorithm, and the General Punctuations block contains explanatory text for each punctuation mark to make sure that not a single character is misunderstood. A list of some character properties together with their description can be found in Table 1.

The Unicode Consortium is quite proactive, as well as responsive to requests, in encoding new characters and redefining character properties. Its Unicode Technical Committee meets quarterly, and tries its best in both keeping stability and backward compatibility, and fixing any existing mistakes. The author has had a pleasing experience with his proposals to the committee.

Character Properties Some normative properties worth special note are *canonical decompositions*, *compatibility decompositions*, and *combining classes*. For instance, Unicode has two ways to encode 'ä', one being the character U+00E4, Unicode Small Letter A With Diaeresis, and the other the sequence of characters ⟨U+0061, U+0308⟩, which is actually a Latin Small Letter A, followed by a Combining Diaeresis. To help applications, Unicode specifies a mechanism for decomposing the former character to the latter sequence, to check equivalence. The situation will get more complicated when you consider the case of double or multiple accents: a Combining Cedilla can be equally followed or preceded by a Combining Tilde, but you can't say that about a Combining Tilde and a Combining Acute Accent, where an interchange will result in different renderings. So, there will be a need for combining classes, which are numbers assigned to each combining character (equal numbers specify non-interchangeable order). The other properties, compatibility decompositions, are decompositions specifying approximate equivalence, like the character U+210E, Planck Constant (h) which is specified to be compatibly equivalent to U+0068, Latin Small Letter H, with only a font difference.

Based on these two ideas, come Unicode *normalization forms* (see Davis and Dürst, 2002). The normalization forms are there to help one do binary checking instead of heavy table lookups. Once the text is in a normalization form such as NFD, in which all precomposed characters are decomposed based on their canonical decompositions, and combining characters sorted based on their combining classes[4], equivalent strings will become equal and it will be good old days again, where you could check string equivalence using the C function `strcmp` or search your files with the UNIX tool `grep`.

In practice, it is Normalization Form C (NFC), a form that re-composes the characters after decomposition, which is the most important form. Having maximum compatibility with legacy character sets, and requiring a simpler rendering logic which eases implementation in portable devices, NFC is referenced frequently in the Character Model for World Wide Web (Dürst et al., 2002) as the normalization form required for all web content. It is also the preferred way for encoding text files and file names in UNIX (except in Mac OS X, where NFD is used). Another form, NFKC, which additionally uses compatibility decompositions, is a requirement of IETF's International Domain Names in Applications (Fältström et al., 2002).

Another important character property, which is absolutely required in the area the author lives, is the *bidirectional category*. These are categories assigned to each character specifying its behavior in mixed right-to-left and left-to-right text, which is common in scripts like Arabic and Hebrew. Characters are divided to classes like *left-to-right*, *right-to-left*, *European number*, *Arabic number*, *common separator*, *white space*, ... which are used in a carefully specified Bidirectional Algorithm (Davis, 2002) to reorder a *logically-ordered* stream of characters to a *visually-ordered* one.

There are also many other character properties, but worth special notice is that breaking almost any of them will make your application non-compliant. You cannot have your Arabic Comma behave as a strictly *right-to-left* character, render two canonically equivalent strings in two different ways, or

[4] The compatibility decompositions will be ignored.

Code Point	The numeric code assigned to the character: ‘2057’ for the Quadruple Prime. This can never be changed.
Character Name	A name only using uppercase Latin letters, digits, hyphen and space. This can never be changed.
General Category	A category describing the general behavior of the character: ‘Sc’ (Symbol, Currency) for the Euro Sign.
Canonical Combining Classes	A class used for ordering combining marks and accents after a base letter: ‘230’ (Above) for Combining Tilde and Combining Grave Accent and ‘202’ (Below Attached) for Combining Cedilla.
Bidirectional Category	Categories specifying the behavior of the character in a bidirectional context: ‘AL’ (Right-to-Left Arabic) for Arabic Letter Beh.
Character Decomposition Mapping	A decomposition of the character to a sequence of others: ‘0075 0302’ (⟨Latin Small Letter U, Combining Circumflex Accent⟩) for Latin Small Letter U With Circumflex, and ‘⟨super⟩ 0054 004D’ for Trade Mark Sign ($^{\text{TM}}$). There are many stability requirements for the character decompositions (Davis and Dürst, 2002).
Decimal Digit Value	A numeric value for digits: ‘3’ for Arabic-Indic Digit Three.
Numeric Value	A numeric value for characters that specify numbers: ‘1/5’ for Vulgar Fraction One Fifth.
Mirrored	Specifies if the character image should be mirrored in text laid out from right to left: ‘Y’ (Yes) for Element Of.
Uppercase Mapping	A one-to-one mapping for converting letters to uppercase: ‘053F’ (Armenian Capital Letter Ken) for Armenian Small Letter Ken (for full case mappings, see Davis, 2001).
Lowercase Mapping	Similar to uppercase mapping, providing lowercase forms: ‘1043E’ (Deseret Small Letter Jee) for Deseret Capital Letter Jee.
Titlecase Mapping	Similar to uppercase mapping, providing titlecase forms: ‘01C8’ (Latin Capital Letter L With Small Letter J for Latin Small Letter Lj).
Arabic Joining Type	Specifies the shaping behavior of an Arabic or Syriac letter: ‘D’ (dual-joining) for Arabic Letter Sheen.
Arabic Joining Group	Specifies a group for each Arabic and Syriac letter, specifying the letter it will be shaped like: ‘SEEN’ for Arabic Letter Sheen.
Line Breaking Property	Specifies how the character behaves relative to line breaking: ‘BA’ (Break Opportunity After) for En Dash (see Freytag, 2002 for more details).
Special Casing Properties	Uppercase, lowercase, and titlecase mapping for languages that handle the case differently: ‘0130’ (Latin Capital Letter I With Dot Above) for Latin Small Letter I in Turkish and Azeri contexts.

Table 1: Some of the various standard properties of Unicode characters.

use some unassigned code points for document interchange, and claim compliance at the same time. The user undoubtedly doesn't like her text to have different meanings in different applications.

Interesting Features These features are some of the interesting ones for typographically and semantically oriented mind sets. Some appear only in later versions of the standard.

- Having different characters for letters that look the same in uppercase but have different lowercase forms. Also, providing different characters for mathematically different forms of letters like the Greek small Phi.
- The ability to recommend for or against automatic ligation in special circumstances, using the characters ZERO WIDTH JOINER and ZERO WIDTH NON-JOINER.
- Specifying *line breaking* properties for characters, to help recommending for or against a line break.
- Specifying a *mirroring* behavior for some characters like the opening parenthesis, whose image should be mirrored in a right-to-left context.
- The ability to encode implicit mathematical semantics, like an invisible times operator or an invisible comma separator, and even a smart DIVISION SLASH for in-line fractions like $^{22}/_{7}$.
- Having characters for almost every symbol required in mathematics typesetting from four combining characters for different forms of `\not` to a character for rare symbols like MATHEMATICAL SANS-SERIF BOLD SMALL XI, and even bracket, brace, and parenthesis fragments, like those available in TeX's `cmex` font.

OpenType

Because Unicode does not provide standard codes for glyphs, but rather is interested only in characters, there is a need for a standard to fill the gap, that is, provide a mechanism to map characters to glyphs in a font. There are three available: Microsoft and Adobe OpenType, Apple AAT, and SIL Graphite (Correll, 2000). Of these, OpenType, a superset of TrueType, has become much more successful, with many outside implementations including some from IBM and the FreeType and GNOME projects. It is becoming the *de facto* standard not because the specification is very clear or technically supreme, but since many high quality fonts exist in the format[5]. Actually, when you get to scripts a little more complex than European ones, say Devanagari or Khmer, OpenType is the only font format in which you will be able to find a couple of usable fonts.

Unlike the AAT and Graphite formats mentioned above, OpenType fonts do not encode the basic visual behavior of the characters in the scripts they support. It's the text layout engine that should know about the script, and the font will only provide the minimum needed information for locating various presentation forms, kerning, positioning accents and marks, and ligating (among others). For example, a font will not include any information about the contextual shaping behavior of U+0649, ARABIC LETTER ALEF MAKSURA, like if it is a right-joining letter or a dual-joining one[6]. It will only specify that a final presentation form of the letter can be found at a certain glyph position.

OpenType introduces new font tables once in a while (even allowing font developers to register some with the specification owners), specifying features like Hebrew mark positioning or Arabic swash forms. This means that the font format is both backward compatible (old engines won't know about advanced features, but can still render the text using the older features available in the font), and extensible (an overlooked feature in a minority script can be requested by anyone and included in the next version of the specification, and everyone will be free to implement or ignore it).

Other Friends

Unicode and OpenType are joined by standards like Adobe PDF (Adobe Systems Incorporated, 2001) which in the latest version provides some mechanisms for having a glyph stream and a Unicode character stream specify a document's visual layout and semantic content in collaboration, W3C Cascading Style Sheets (Bos et al., 1998; Suignard and Lilley, 2001) that includes mechanisms to guide automatic selection of fonts for Unicode characters from various scripts that appear in a single text document, and MathML, with almost all of its symbols now in Unicode, which will happily let you cut and paste mathematical formulas between different applications.

[5] For example, Adobe recently re-released all of their professional fonts in the OpenType format, abandoning both PostScript Type 1 and Multiple Master for new fonts.

[6] Actually, the joining class of this letter was changed between Unicode versions 3.0 and 3.0.1.

Many other standards have also emerged, both using Unicode's power in supporting technical symbols and minority scripts, and learning from its successful unifying experience. Fortunately, these standards are not only on paper, but they are followed and being implemented carefully by both major software houses and Open Source and Free Software communities. The author is specially interested in the linux-utf8 mailing list, where discussions about implementing Unicode-related technologies in GNU, Linux and UNIX platforms happen (subscription information is available at `http://www.cl.cam.ac.uk/~mgk25/unicode.html#lists`).

Where We Stand

In this picture, TeX and friends are left far behind the majority of the world. We still mostly use our own font formats, METAFONT, PostScript Type 1, TFM, and PK instead of OpenType (for outline fonts) or pcf and bdf (for bitmap fonts)[7]; the TeX output format DVI is still in very wide use, whereas the rest of the world primarily uses PDF; and almost all TeX friends have their own traditional command sequences with a very loose syntax, in contrast with a world community united on XML. In short, they just don't fit!

This should not be compared to the beginning years, when the TeX community needed to invent formats for technologies that were becoming public for the first time, and had the chance of making them *de facto* standards. Many components can (and should) now be replaced with their widely recognized and recommended equivalents like XSL Formatting Objects (Adler et al., 2001), to have the luxury of high quality typography once again. Contrary to many who insist that they are dead horses not worth more beatings, the author believes that TeX and friends have the ability of reaching many new users if they get modified to support current technologies.

New Requirements for Omega

The author considers Omega as a possible candidate to advance in the field and fill the existing gap. Actually, it is the only candidate among friends like ε-TeX and pdfTeX, which are more typographically oriented. Omega has already passed the hassle of implementing sixteen-bit fonts, pre- and post-processing text filters needed for rendering the so-called complex scripts, and even some MathML. But unfortunately it's not stable, lacks an active development team, and has a rather more academic orientation than desirable for such a project. The worst point of all is that it is being developed in a cathedral model (see Raymond, 2001), which makes it very hard to reach the above goals.

To make Omega usable in the current working environment, the following requirements come to mind:

- Opening the development of Omega, not only accepting contributions from the public, but also making them active in the development process. This will need a few central people at the beginning, with enough time at their hand to sketch and implement a working mechanism for future development.
- Accepting the international standards as they are, and trying to be compliant as much as possible: any problem in standards like Unicode should be taken to the Unicode authors themselves, instead of trying to fix them locally[8]. This will help reach consistency with many other existing tools, or those who may be developed in the future.
- Implementing PDF output (not necessarily a merge with pdfTeX), including Unicode character streams.
- Implementing native OpenType support. Some first milestone for achieving the goal may be providing a tool to convert OpenType tables to ΩTP processes. Also, the current Omega fonts should be converted to the OpenType format[9]. Apart from making Omega able to use almost all of the fonts in the wild, this will be a contribution to the Open Source community who lack good printing engines for texts in non-European scripts.
- Supporting XML and MathML as both input and output formats. Fortunately, some of this has already been implemented.
- Closely tracking new features of Unicode, so as to stay current with the rest of the world. Omega may even be able to act faster than its competitors in this field, especially if it starts to follow the *bazaar* model of development.

[7] Even tools like ttf2pk, which let TeX use TrueType fonts, are not in active development and are based on a frozen branch of the FreeType font engine

[8] The author believes that incorporating a fix which will definitely happen in a yet unpublished version of a standard should be allowed.

[9] Some 'Free UCS Outline Fonts', UCS standing for Universal Character Set which is the alternate name of Unicode, are under development based on existing free outlines including those from Omega. Latest information is available at `http://savannah.gnu.org/projects/freefont`.

Acknowledgments

The author wishes to thank Markus Kuhn and Sebastian Rahtz for initially raising some of the issues discussed in this paper, Martin Dürst, Mark Davis, and many other internationalization evangelists who have helped him understand the development mechanisms of many internationalization-related standards, and participate in their development, and also Karl Berry, for editorial assistance. Final thanks should be given to John Plaice and Yannis Haralambous who created Omega in the first place.

References

Adler, Sharon, A. Berglund, J. Caruso, S. Deach, T. Graham, P. Grosso, E. Gutentag, A. Milowski, S. Parnell, J. Richman, and S. Zilles. "Extensible Stylesheet Language (XSL), Version 1.0". W3C Recommendation, World Wide Web Consortium, 2001. Available from `http://www.w3.org/TR/xsl/`.

Adobe Systems Incorporated. *PDF Reference, Adobe Portable Document Format Version 1.4*. Addison-Wesley, third edition, 2001. Also available in electronic format from `http://partners.adobe.com/asn/developer/acrosdk/docs/filefmtspecs/PDFRe%ference.pdf`.

Adobe Systems Incorporated and Microsoft Corporation. "OpenType Specification Version 1.3". 2001. Available from `http://partners.adobe.com/asn/developer/opentype/main.html` and `http://www.microsoft.com/typography/otspec/default.htm`.

Bos, Bert, H. W. Lie, C. Lilley, and I. Jacobs. "Cascading Style Sheets, level 2". W3C Recommendation, World Wide Web Consortium, 1998. Available from `http://www.w3.org/TR/REC-CSS2`.

Carlisle, David, P. Ion, R. Miner, and N. Poppelier. "Mathematical Markup Language (MathML) Version 2.0". W3C Recommendation, World Wide Web Consortium, 2001. Available from `http://www.w3.org/TR/MathML2`.

Correll, Sharon. "Graphite: An Extensible Rendering Engine for Complex Writing Systems". Technical report, SIL International, 2000. Available from `http://graphite.sil.org/pdf/IUC17_paper.pdf`.

Davis, Mark. "Case Mappings". Unicode Standard Annex #21, The Unicode Consortium, 2001. Available from `http://www.unicode.org/unicode/reports/tr21`.

Davis, Mark. "The Bidirectional Algorithm". Unicode Standard Annex #9, The Unicode Consortium, 2002. Available from `http://www.unicode.org/unicode/reports/tr9/`.

Davis, Mark and M. Dürst. "Unicode Normalization Forms". Unicode Standard Annex #15, The Unicode Consortium, 2002. Available from `http://www.unicode.org/unicode/reports/tr15/`.

Dürst, Martin J., F. Yergeau, R. Ishida, M. Wolf, A. Freytag, and T. Texin. "Character Model for the World Wide Web 1.0". W3C Working Draft, World Wide Web Consortium, 2002. Available from `http://www.w3.org/TR/charmod`.

Fältström, Patrick, P. Hoffman, and A. M. Costello. "Internationalizing Domain Names in Applications (IDNA)". Internet Draft, Internet Engineering Task Force, 2002. Available from `http://www.ietf.org/internet-drafts/draft-ietf-idn-idna-10.txt`.

Freytag, Asmus. "Line Breaking Properties". Unicode Standard Annex #14, The Unicode Consortium, 2002. Available from `http://www.unicode.org/unicode/reports/tr14`.

Raymond, Eric S. *The Cathedral and the Bazaar, Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly, revised edition, 2001. Also available in electronic format at `http://tuxedo.org/~esr/writings/cathedral-bazaar/`.

Suignard, Michel and C. Lilley. "CSS3 module: Fonts". W3C Working Draft, World Wide Web Consortium, 2001. Available from `http://www.w3.org/TR/css3-fonts`.

The Unicode Consortium. *The Unicode Standard, Version 3.0*. Addison-Wesley, 2000. Also available in electronic format from `http://partners.adobe.com/asn/developer/acrosdk/docs/filefmtspecs/PDFReference.pdf`.

Unicode Editorial Committee. "Unicode 3.1". Unicode Standard Annex #27, The Unicode Consortium, 2001. Available from `http://www.unicode.org/unicode/reports/tr27/`.

Unicode Editorial Committee. "Unicode 3.2". Unicode Standard Annex #28, The Unicode Consortium, 2002. Available from `http://www.unicode.org/unicode/reports/tr28/`.