

# **Unicode**

---

A progress report on its impact on Asian  
languages

---

---

## Introduction

In 1997 the NCOLR IT Subcommittee explored the possibility of holding a conference on Unicode, the new miracle encoding system that promised a unique identifier for every character, no matter what the language, no matter what the program, no matter what the platform. The idea for the conference seemed to be overtaken by events as software companies swiftly moved towards Unicode compliance. NCOLR Committee members felt that the challenges presented by input, rendering, and data transfer in non-roman scripts would soon be a thing of the past and the idea for the Unicode conference was dropped.

The six years following the discussions about a conference have seen considerable progress in Unicode implementation but have also revealed that the challenges presented by input and rendering of non-roman scripts have not gone away. It therefore seemed appropriate for the NCOLR IT Sub-committee to summarize the history of Unicode, to explore how far it is possible to successfully implement Unicode for Asian scripts and to assess how far there is still to go before the challenges presented by input, rendering and data transfer in non-roman scripts really do become a thing of the past.

---

## Background to Unicode

The Unicode encoding system began as a project in late 1987 after discussions between engineers from Apple and Xerox. The name Unicode was coined by Joe Becker of the Xerox group from unique, universal, and uniform character encoding. Unicode sought to address the interoperability problems caused by the multiple encoding systems in use at the time.

Fundamentally, computers just deal with numbers. They store letters and other characters by assigning a number for each one. Before Unicode was invented, there were hundreds of different encoding systems for assigning these numbers. No single encoding could contain enough characters: for example, the European Union alone requires several different encodings to cover all its languages. Even for a single language like English no single encoding was adequate for all the letters, punctuation, and technical symbols in common use.

These encoding systems also conflict with one another. That is, two encodings can use the same number for two *different* characters, or use different numbers for the *same* character. Any given computer (especially servers) needs to support many different encodings; yet whenever data is passed between different encodings or platforms, that data always runs the risk of corruption.<sup>1</sup>

---

<sup>1</sup> <http://www.unicode.org/standard/WhatsUnicode.html> [website last accessed 27/10/03]

---

In 1988, work began on the first database for the Unicode names and mappings and in early 1989, the scope of the Unicode working group was extended to include a number of other companies and Karen Smith-Yoshimura and Joan Aliprand of the Research Libraries Group.

At the same time as the Unicode project was underway in the US, work on a unified character encoding system, known as ISO 10646, was also taking place in Europe. The two standards were incompatible, so the translation between the codes was impossible. A series of politically sensitive talks took place in 1991 between advocates of ISO 10646 and Unicode, which enabled the merger of the two standards and prevented the launch of two competing encoding systems.

The initial Unicode project team developed into the Unicode Consortium, which is a non-profit organization founded to develop, extend and promote use of the Unicode Standard. The membership of the consortium represents a broad spectrum of corporations and organizations in the computer and information processing industry. The consortium is supported financially solely through membership dues and is open to organizations and individuals anywhere in the world who support the Unicode Standard and wish to assist in its extension and implementation.

The Unicode Technical Committee (UTC) is the primary decision making body within the Unicode Consortium. It meets quarterly to consider proposals for script additions or changes, and to decide on recommendations to other standards bodies. The UTC has four sub-committees; the Bidi subcommittee, which focuses on Unicode's support for bi-directional scripts; the East Asian subcommittee, which addresses all issues pertaining to the set of character-based Far East writing systems and languages; the Editorial subcommittee, which focuses on the production of officially published documents from the Consortium; and the Scripts and Symbols subcommittee, which acts as a collection point for proposed additional characters from any script, and proposed additional symbols.

The fourth version of the Unicode Standard was published this year.<sup>2</sup> An online version in PDF format is available from the Unicode consortium website<sup>3</sup> but, in order to protect sales of the book, the printing function has been disabled.

---

<sup>2</sup> The Unicode Consortium. The Unicode Standard, Version 4.0.0, defined by: *The Unicode Standard, Version 4.0* (Boston, MA, Addison-Wesley, 2003. ISBN 0-321-18578-1)

<sup>3</sup> <http://www.unicode.org/versions/Unicode4.0.0/> [website last accessed 29/10/03]

---

# Bits, Bytes, and Encoding Systems

## How computers store text

### A brief history of encoding

#### What is encoding?

As mentioned earlier, computers just deal with numbers. In order to store letters, it was necessary to convert them into numerical codes, so that the text could be stored and reproduced. The rendering (drawn shape) of the text element (character) became divorced from “what it represented.” This process of allotting a numerical code to a character is called encoding. Character codings were used to represent the fact of a character being in a particular place in a text without saying anything about its shape, size, colour, etc.

Computers structure their memory in chunks ‘bytes’. Each chunk or byte is made up of a number of bits. A **bit** or **B**inary **digiT** is an electronic impulse that can be represented by two states “on” or “off”, also written as “1” or “0”.

In early computers one chunk of byte of memory was used to represent each character. Different computers used different numbers of bits to make up a byte, however, and this made it difficult to exchange of information between machines.

#### ASCII

The early computer industry eventually agreed on a standard sized byte of 7-bits. The 7-bit byte allowed for 128 characters to be encoded, which was sufficient to handle upper and lower case English, punctuation, numbers and some control codes. This became the ASCII standard, which stands for American Standard Code for Information Interchange.

ASCII encoding worked well for some years, as computing was mainly an English language preserve and text based computing was relatively unimportant.

Even with the advent of ANSI and Unicode, e-mail messages are still commonly sent in 7-bit encoding because 8-bit or 16-bit encoded characters may not pass correctly through some gateways.

#### ANSI

The advent of microcomputers with improved memory capacity in the 1980’s led to the industry wide adoption of the 8-bit byte. An 8-bit byte system offered 256 locations that could be used for characters. As bytes had room for up to 8-bits, there were now 256 locations that could be used for characters. People realized that the code positions 128-255 were “spare” and could be used for their own purposes. The character set known as OEM, which was used on the IBM-PC, provided some accented characters for European languages and a number of line drawing characters. As soon as IBM-PCs came to be sold outside of America, all sorts of OEM character sets, using the top 128 characters, were

developed. For example, on some PC's the character code 130 would display as é, but on computers sold in Israel it was the Hebrew letter Gimel (ג). In many cases, such as Russian, there was no agreement on the assignment of the upper 128 characters, so it was not possible to reliably interchange documents in the same language.

Eventually the ANSI (American National Standards Institute) standard developed from this chaotic free-for-all. In the ANSI standard, everybody agreed on a common set of characters below 128, which was virtually the same as that used in ASCII encoding. Characters above 128 were then handled differently according to the part of the world in which the PC was being used. These different encodings above position 128 were called code pages. Hebrew used code page 862, for instance, while Greek used code page 737.

The ANSI code tables offered an important step forward in internationalization, as there was now an agreed standard that allowed different alphabets to be encoded. As the same set of high end encoding values were being used for different languages, however, it was not possible to use different languages on the same computer without a custom program.

### **East Asian scripts**

Far Eastern scripts had too many letters to fit into an 8-bit encoding system so the Double-byte Character Set was developed for use in languages such as Japanese. This is a rather messy system in which some letters are stored in one byte but others are stored in two. DBCS character handling requires detailed changes in the character-processing algorithms in order for applications to determine whether a given character is a 2- or 1-byte character.

For Chinese, two encoding systems emerged, one for use with full form or traditional Chinese, as used in Taiwan (about 13,500 glyphs), and one for use with simplified Chinese, as used in mainland China (about 7000 glyphs).

Meanwhile, in America, the Research Libraries Group developed an encoding system known as the East Asian Character Code (EACC) for use with Chinese, Japanese and Korean. EACC was approved as an American National Standard, ANSI/NISO Z39.64, in 1989 for use in MARC 21 records. The EACC repertoire contained 15,728 characters as of May 2001.

### **Unicode**

Unicode is commonly thought to be a 16-bit encoding system, where each character takes 16-bits, offering the possibility of 65,536 possible characters. In fact, it is theoretical construct rather than a practical encoding system with no real limit to the number of letters that it can define. Indeed, it has already exceeded the total number of characters available in 16-bit encoding.

In an encoding system each letter maps to some bits, which can be stored on a disk or in memory. In Unicode, a letter maps to something called a *code point*, which is a theoretical concept. How that code point is represented in computer memory is a separate issue.

In Unicode, the theory is that each letter is a platonic ideal; **A** is considered to be different from **B** but the same as **A** or *A* which are just different renderings of the same letter.

The Unicode consortium has made numerous decisions about whether a letter in a certain alphabet is separate character or simply an alternative rendering of another character. To

the outsider the reasoning behind some of these decisions does not always seem apparent. In the Chinese code tables, every graphic variant of a letter merits a separate code point. At the other extreme, Indic consonants, which change their shape when combined with other consonants, are considered to be the same character with the same code point, regardless of the context in which they occur.

Every platonic letter in the alphabet is assigned a hexadecimal number with the prefix U+, which stands for Unicode. For example, the Arabic letter Ain has the code point U+FEC9. This code point, however, does not say anything about how it should be stored in a computer's memory. In order to make practical use of the Unicode code points they need to be encoded.

### **Unicode Encoding**

The early Unicode encoding systems adopted two bytes (16 bits), called **UTF-16**, Unicode Transformation Format-16 (because it has 16 bits). This two byte encoding has several disadvantages. Compared with 8-bit encoding, it doubles the amount of storage space needed in computer memory. English text rarely uses code points above U+00FF so much of the extra space is used to store zero values and, in a world still dominated by the use of English language for computing, this was regarded as wasteful. A further disadvantage was the fact that the world was already full of applications encoded in ANSI and DCBS, which created a huge issue of data migration.

The invention of **UTF-8** addressed these two problems and it has quickly overtaken UTF-16 in popularity. In UTF-8 every code point from 0-127 is stored in a single 8-bit bite. English text, therefore, looks exactly the same as it did in ASCII and can therefore be correctly interpreted by programmes that only work in ASCII. Code points above 127, which include the Asian scripts, are stored in 2 or more bytes.

There is also an encoding called **UTF-7**, which is extremely similar to UTF-8 and designed to circumvent e-mail gateways that only accept 7-bit bytes by ensuring that the high bit will always be zero.

As Unicode code points are theoretical constructs, they can equally well be encoded in any of the older encoding systems, it is just that the older encoding standards are not able to store all of the code points. If there is no equivalent for the Unicode point in the encoding system then it is converted into a question mark.

---

## **Characters, Glyphs & Fonts**

### How computers display text

#### **What are glyphs & fonts?**

As discussed in the earlier section, computers store letters as numbers. If these encoded letters are to be displayed on the computer screen they have to be translated into the shapes. The exact shape by which a character is represented is called a **glyph**.

A **font** is a collection of glyphs, all of similar design, that constitute one way to represent the characters of one or more languages. Fonts usually have some element of design consistency, such as the shape of the ovals (known as the **counter**), the design of the stem, stroke thickness, or the use of **serifs**, which are the fine lines stemming from the upper and lower ends of the main strokes of a letter.

Unicode determines code points to be used in encoding and is not a type of font. A "Unicode font" can be any of a number of types, such as True Type, Open Type, PostScript etc.

### **Sources of Unicode fonts**

A number of Unicode fonts for Asian languages have been developed and one of the best listings of what is available can be found on Alan Wood's list of Unicode resources.<sup>4</sup>

---

## **Some Unicode font problems**

### **One letter but many glyphs**

In European scripts, each Unicode character can be represented by a single glyph. However, things are far more complicated for handwritten cursive scripts such as Arabic, Syriac and the various Indic scripts. In an Arabic or Indic font, the shape of the glyph depends not only on the character that it represents, but also on its neighbouring characters. Sometimes, different glyphs have to be used depending on the character appearing at the beginning, middle, or end of a word, and often certain entire sequences of characters have to be represented by a special ligature glyph. A very simple form of that is used in Latin fine typography in the form of the "fi" and "fl" ligatures, but in Indic scripts, the situation is far more extreme, and the number of glyphs is often several times the number of characters.

The Unicode standard does contain encoding ranges for a simple scheme of Arabic glyphs, the "Arabic Presentation Forms". This was possible, because for Arabic there is a reasonably good consensus among font designers on how many glyphs are actually necessary for proper rendering of Arabic text, even though some argue that for really high-quality typesetting the Unicode collection of Arabic presentation forms is not sufficient. For Indic scripts on the other hand, there seems no consensus among font designers, which glyphs are actually necessary as this can vary significantly across different font styles. Therefore, an Indic font is always a proprietary non-standardized collection of glyphs together with a mapping table that defines how sequences of standard Unicode characters have to be transformed into sequences of non-standard Indic glyphs from this particular font, before the text can be displayed. The difficulties confronting those using Indic scripts will be discussed in more detail under Indic languages.

---

<sup>4</sup> <http://www.alanwood.net/unicode/fontsbyrange.html> [website last accessed 12/1//03]

## One letter but many code points

In Chinese, the decision to create a separate code point for every graphic variant means identical words written with slightly different glyphs have different encodings. This gives a great deal of flexibility when using Unicode fonts for text display but when such words are entered into a database they are indexed in different places.

## The problem of aligning combining diacritical markings

Unicode offers two types of diacritical markings, pre-composed characters in which the diacritical marking is an integral part of the glyph and the combining diacritical marking, where the diacritical markings are on a separate glyph from the character they modify and are displayed in the same space as the character that they modify. Some languages have considerable numbers of combining characters, e.g. Hebrew has 48. Unfortunately, combining diacritics in TrueType fonts (the most commonly available types of Unicode fonts) do not tend to line up correctly over or under the base character. At best, the result is a rather crude appearance and at worst it can produce confusion between different letters (see under Arabic).

---

# Solutions for the multiple glyph/combining diacritical problems

## The Private Use Area

Some font designers have approached these problems through the utilization of Unicode's **Private Use Area (PUA)**. The Private Use Areas is a block of code points within Unicode, which is reserved for characters which will never have code points assigned to them by the Unicode standard. Fonts, such as Alphetum<sup>5</sup>, have solved the difficulty of the need for different glyphs in the Indic script Devanagari by placing characters for the half forms of consonants in into the Unicode Private Use Area.

Unfortunately this approach resurrects all the problems that Unicode was designed to solve as there is no consensus about how to use the PUA., even between fonts for the same language.<sup>6</sup>

## TrueType Open fonts

The solution to problems caused by glyphs and combining diacritical markings has been approached by Microsoft and Adobe through the development of the TrueType Open standard, which is an extension to the TrueType font standard. TrueType Open fonts contain additional information that extends the capabilities of the fonts to support non-roman scripts. The TrueType standard offers the following features.<sup>7</sup>

- TrueType Open fonts allow a rich mapping between characters and glyphs, which supports ligatures, positional forms, alternates, and other substitutions.

---

<sup>5</sup> <http://quindo.cnice.mecd.es/~jmag0042/alphaeng.html> [web page last accessed 13/11/03]

<sup>6</sup> Alphetum (Version 5.10) Users Manual, p. 7

<sup>7</sup> <http://www.microsoft.com/typography/default.mspx> [web page last accessed 13/11/03]

- TrueType Open fonts include information to support features for two-dimensional positioning and glyph attachment.
- TrueType Open fonts contain explicit script and language information, so a text-processing application can adjust its behavior accordingly.
- TrueType Open fonts have an open format that allows font developers to define their own typographical features.

A TrueType font is a collection of several tables that contain different types of data: glyph outlines, metrics (horizontal spacing information for each glyph), bitmaps, mapping information, and much more. TrueType Open fonts contain all this basic information, plus five additional tables containing information for advanced typography, including tables that provide information about possible glyph substitutions, scripts and language systems.

TrueType Open fonts can be used by operating systems and applications that support TrueType but do not implement TrueType Open functionality. The main problem with the OpenType solution is that creation of TrueType Open fonts in Asian languages is still in its infancy and, even where such fonts do exist only a few programs as yet support the advanced features of OpenType. Applications without OpenType support can still use OpenType fonts but do not have access to some features and can not access all of the glyphs in the font.

In the table of applications which support for OpenType fonts on the myfonts.com website<sup>8</sup> only Adobe's InDesign and Adobe's Photoshop are listed as having support for the advanced features necessary to render fonts in Asian scripts successfully.

---

## Databases, Web-Development and Unicode

### Support for Unicode in Bibliographic Databases

Most of the major software companies that sell large bibliographic databases for libraries now support Unicode, mostly with UTF-8 encoding. Most have adopted Arial MS Unicode font for non-roman scripts. There are usually display and searching facilities for CJK languages, though not all are satisfactory.<sup>9</sup> Arabic and Hebrew script entries can also be searched and displayed but with certain difficulties noted under the sections on Arabic and Hebrew. With the possible exception of VTLS<sup>10</sup>, Indic scripts are not supported by any of the major database applications. Sales teams are confident that their databases can handle True-Type Open fonts, should they become available, but the lack of support for the extended features in other software suggests that implementation is not trivial and the confidence of the sales teams could well be misplaced.

---

<sup>8</sup> <http://www.myfonts.com/info/opentype-support-in-applications/> [website last accessed 13/11/03]

<sup>9</sup> Cambridge University's Voyager system still has unresolved problems in display of CJK languages.

<sup>10</sup> VTLS Library systems do have customers in India but IT-Subcommittee members have been unable to verify whether they are using Indic scripts

Open-source databases have lagged behind commercial applications in supporting Unicode. The first version of MySQL to offer extensive support for Unicode in UTF-8 encoding was only released this year, and, as with all new releases, has contained some bugs.

## **Web-Editors**

Support for Unicode encoding in web-authoring applications is still patchy. In Dreamweaver MX, for example, a Japanese font set in a stylesheet (.css file) did not display correctly in the editor, so it was necessary to use the deprecated <font> tag whilst inputting text. The very latest version of Dreamweaver (Dreamweaver MX 2004) has fixed this bug but introduced other problems. It is slow to start, uses huge amounts of memory and crashes frequently. It still does not handle right-to-left languages e.g. Arabic, Hebrew, Persian and Urdu. FrontPage, although in many ways a less sophisticated editor than Dreamweaver, is still the best option for those developing webpages that require right-to-left scripts.

---

# **Arabic, Persian & Unicode**

## **Arabic on the Web**

Since the introduction of Word 2000 and Unicode fonts such as Arial MS Unicode, Arabic on the web seems to be on the increase. There are now several web-based Arabic email services, and most services and commodities (from newspapers to wedding services to online stores) are now available in Arabic on the web. Google has an Arabic interface, and there are other local search engines and hundreds of gateways.

OCLC First Search now offers Arabic script through the option of selecting "show vernacular." Other sites that use Arial to display large quantities of text are [www.neelwafurat.com](http://www.neelwafurat.com); and [www.alwaraq.com](http://www.alwaraq.com). The former is an online bookshop, and the latter a vast online library.

One of the most popular fonts in use for Arabic script on the web is Arial MS Unicode,. It is one of the most comprehensive Unicode fonts presently available for Arabic-script languages, but it has its deficiencies.

The main problems are:

- Diacritic dots are too small when the typeface is bold. This makes it difficult to distinguish certain letters.
- When underlined, diacritic dots below the letters actually merge with the line and disappear. This renders some words illegible (although they can usually be guessed).

- In normal type, at 10-12 pt, certain letters are difficult to distinguish, either because the dots are not centered, or because the distance between diacritic dot and carrier is too small (as in the case of the nun, which sometimes appears almost like a lam)
- OCLC appears to have problems with the Persian characters cha (0687) and gaf (06AF), both of which are provided for in Unicode, and both of which are available in Arial Unicode MS. This is curious: other Persian sites display these characters without any problem.
- The font is strictly linear, i.e every new character appears to the left of the preceding one. In writing as in typesetting, certain character sequences melt into traditional ligatures (the most common ones involve the letters mim and those of the Ha' group). Although this does not affect legibility, it does make the text look odd, particularly with names like Muhammad, which are never written without ligatures.

The other fonts which are used for Arabic on the web are TNR, Arabic Transparent, and Simplified Arabic Fixed. TNR is a very comprehensive Unicode font similar to Arial MS Unicode. It is legible, and aesthetically more appealing than Arial but does not seem to have the same popularity.

## **CJK scripts & Unicode**

### **Chinese**

#### **Chinese Simplified & Traditional**

The two encoding systems which emerged for Chinese, Traditional, as used in Taiwan, and Simplified, as used in mainland China, were both given sets of Unicode code points. In addition, variants of traditional Chinese characters were also given separate code points in Unicode. While this facilitates the transition to Unicode code points, for these two systems, it presents problems if simplified and traditional Chinese are used in the same database.

Since the same character is indexed in two different places, it is necessary to make two searches in order to retrieve all the entries. This can only be streamlined if cataloguing standards involve some kind of normalization process during the entry of the bibliographic details, so that the character is always entered in the same form, or if the database has some kind of mapping mechanism that allows for the retrieval of all the different forms of a character with one search.

The electronic version of SI BU CONG KAN, a collection of classic Chinese works, is an example of a Unicode application which uses mapping to enable retrieval of the different forms of character. The full-text electronic version of SI KU QUAN SHU (Published by Digital Heritage in Hong Kong) is also Unicode compliant. Users report that the small number of characters outside the Unicode character set does not cause significant problems.

## EACC

The East Asian Character Set Task Force was formed by MARBI in 1997 to establish mappings between EACC and Unicode. The work of the Task Force focused specifically on reviewing mappings of East Asian characters already done by the Unicode Consortium, identifying characters missing from Unicode, establishing mappings for Korean hangul, Japanese kana, CJK punctuation and component characters, and working out a solution for mapping duplicate and variant ideographic characters. RLG now uses MARC 21/Unicode mappings to display original scripts and to support original script searching.

It seems that mapping, however, has involved use of the PUA<sup>11</sup>, with all its attendant drawbacks.

## Japanese

NACSIS data, the source of derived records for the U.K.'s Union Catalogue uses a Unicode encoding (UTF-8) and a Unicode font designed by NACSIS, which is distributed as freeware. The Union Catalogue can be found in two versions <http://bodley24.bodley.ox.ac.uk/cgi-bin/acwww25/maske.pl?db=gbjpn> (hosted by the Bod. in Allegro format) and at <http://juc.lib.cam.ac.uk/> (hosted by Cambridge Univ.).

The Unihan website <http://www.unicode.org/charts/unihan.html> is a useful dayabes for those who need to include obscure characters in word-processed documents. Once located on the database characters can then be cut and pasted into word-processed documents.

---

# Hebrew & Unicode

## Issues in the Representation of Pointed Hebrew in Unicode

Unicode does not provide sufficient code points in order to deal adequately with pointed texts and there are suggestions that the standard may need to define additional characters.

The Hebrew block of the Unicode Standard) is intended to include all of the characters needed for proper representation of Hebrew texts from all periods of the Hebrew language, including fully pointed and cantillated ancient texts such as that of the Hebrew Bible. It is also intended to cover other languages written in Hebrew script, including Aramaic as used in biblical and other religious texts: as well as Yiddish and a few other modern languages.

In practice there are a number of issues and minor deficiencies in the Hebrew block as currently defined, in version 4.0 of the Unicode Standard), which affect its usefulness for representation of pointed Hebrew texts and of Hebrew script texts in some other languages. Some of these simply

---

<sup>11</sup> <http://www.loc.gov/marc/specifications/spechareacc.html> [website last accessed 17/11/03]

require clarification and agreed guidelines for implementers. Others require further discussion and decision, and possibly additions to the Unicode standard or other action by the Unicode Technical Committee<sup>12</sup>.

TrueType Unicode fonts display combining diacritical markings very poorly. The David and Miriam fonts perform better than Arial MS Unicode but the problem will only really be solved with the ready availability of OpenType fonts and applications that can interpret the extra tables included in this type of font.

Ezra SIL fonts do offer a freeware package of two OpenType Hebrew fonts that are designed to work with any OpenType application and are compliant with Unicode 3.2.<sup>13</sup>

---

## South & South-east Asian scripts & Unicode

### ISCII/ Unicode divergences

India has its own standard for character encoding for Indian languages that originate from Brahmi script, known as ISCII (the Indian Standard Code for information exchange). ISCII was evolved by a standardization committee under the Department of Electronics during 1986-88, and adopted by the Bureau of Indian Standards (BIS) in 1991. Unlike Unicode, ISCII is an 8-bit encoding that uses escape sequences to announce the particular Indic script represented by a following coded character sequence. There are several other differences between ISCII and Unicode. The Government of India is a member of the Unicode Consortium, and has been engaged in a dialogue with the UTC about additional characters in the Indic blocks and improvements to the textual descriptions and annotations. As yet, however, Unicode has not been embraced wholeheartedly in India.

### Necessity for OpenType fonts

In an Indic font, the shape of the glyph depends not only on the character that it represents, but also on its neighbouring characters and often certain entire sequences of characters have to be represented by a special ligature glyph. As Unicode only assigns code points to base glyphs, all other glyphs have to be a function of the font.

For example, Gujarati would contain glyphs for the allocated code points in the range: U+0A80 - U+0AFF. In addition to these, the font would have to have: (a) glyphs for conjuncts; (b) variants for vowel signs (matras), vowel modifiers (Chandrabindu, Anuswar), the consonant modifier (Nukta); (c) digits and any appropriate punctuation marks (perhaps some that are appropriate from the Latin ranges). A Glyph Substitution table would be necessary for the font to offer (a) and (b) and Glyph Positioning table would also be needed for achieving the minimal required mark positioning for the script.

---

<sup>12</sup> Kirk, Peter. *Issues in the representation of pointed Hebrew in Unicode*. 3<sup>rd</sup> Draft August 2003. <http://www.gaya.org/academic/hebrew/Issues-Hebrew-Unicode.html> [website last accessed 17/11/03]

<sup>13</sup> <http://www.sil.org/computing/fonts/silhebruni/> [website last accessed 17/11/03]

Such extra tables are a feature of OpenType fonts. Microsoft has made two OpenType Indic script fonts with TrueType outlines available with Windows 2000. They are Mangal.ttf (for Hindi) and Latha.ttf (for Tamil). Unfortunately, as discussed earlier, very few applications are capable of interpreting the Glyph Substitution and Positioning Tables so these fonts are of very little use at present.

### **OpenType fonts and databases**

Database support for OpenType fonts is a complex issue since the additional tables would play a part in indexing and searching. As these tables are not yet supported by standard word processors like Word, it may be some time before database technology is able to correctly store and interpret bibliographic data which uses OpenType fonts.

---

## **Conclusions**

### **An essential building block but not the final answer**

In 1997 it seemed that Unicode offered all the answers to those working with non-roman scripts. As a better understanding of what Unicode actually is has emerged, it is clear that it is only a building block and there is still some way to travel before obstacles to input and rendering of non-roman scripts in computer applications are finally overcome.

Up until now, librarians have focused on the issue of Unicode compliance, but to get the functionality that we really require, we have to start asking questions about fonts and how well they are interpreted by the applications in which we wish to use them.