

Enabling Pakistani Languages through Unicode

Written for Microsoft by Abdul-Majid Bhurgri

Introduction

We will begin this article with a keen and relevant observation made by two leading experts in the field of information technology from Pakistan:

The benefits from Information Technology (IT) revolution cannot be reaped unless masses use it, which is not possible unless computing is possible in a language that is understood by the masses.ⁱ

This perceptive remark defines the very goal that the people of regions like Pakistan must achieve if they wish to make optimum use of the Information Technology. This also lays down the condition precedent for such use. This being so, the corollary questions arise: How to attain this goal and make computing in the regional languages possible? What are the various choices? And of these, which ones are efficient and effective enough to be of any practical help? The purpose of this article is to discuss the related issues objectively, explore various solutions, and assist in the selection and implementation of the better ones. The discussion will however be kept to the Arabic script languages of the region.

Information Technology

We are living in an era of Information Technology and some of the major issues of the era relate as to how IT interacts with languages. Written language has been a basic mode of human communication during the modern as well as medieval times. Language is the medium and the vehicle used by IT to promote itself and advance its objectives, and in turn the languages also get enriched and more widely used as means of written communication. We could say that the information technology and languages have in a sense become very closely linked and interdependent.

That being the case, the issue of effective use of information technology is closely linked with and dependent upon the extent to which the languages of a region are implemented in the technology. People of a region can make advantageous use of the technology only if their native languages can be used in its environment.

English has no doubt evolved into what could rightly be called lingua franca or the language of the world as well as of the IT era. But no language can replace any other language in the normal course of things. The bond of a language with people and their culture and milieu is too deep to be erased easily. Just as variety of languages in the real world adds to its beauty, in the same way presence of various languages in the virtual world, created by information technology, enhances its beauty as well as utility.

Here it is that the interests of vendors of IT and of people of a region coincide and supplement each other. The vendors, by providing support for languages of the region, find new markets for their products; while the people of the region, by being able to use the technology in their native language, make optimal use of it.

Languages of Pakistan

Let us take a stock of Arabic script languages of Pakistan. Some of the major languages are listed in following table along with number of people in Pakistan speaking the language as their mother tongue.¹¹

Language	Number of Speakers
Balochi	5,685,000
Balti	270,000
Brahui	2,000,000
Farsi	1,000,000
Hindko	2,500,000
Kashmiri	105,000
Khowar	223,000
Parkari	250,000
Pashto	11,100,000
Punjabi	30 to 45 m.
Saraiki	15 to 30 m.
Sindhi	17,000,000
Urdu	10,700,000

Table 1

Major spoken languages of Pakistan are: Punjabi, Saraiki, Sindhi, Pashto, Urdu, Balochi, Hindko and Brahui. Of these, only Urdu, Sindhi, and Pashto have a standardized alphabet. There are very few written works available in these other languages. Speakers of these languages, if they ever need to write in their language, use the alphabet of some other major language (usually Urdu or Sindhi) in which they have been formally educated. For Punjabi, mostly Urdu alphabet and writing style is used because most of the Punjabis have received their schooling in Urdu. For Saraiki, Urdu as well as Sindhi alphabet is used because Saraiki is spoken in Punjab as well as Sindh. Balochi also does not have any standardized alphabet. Mostly Urdu, sometimes Farsi, and occasionally Sindhi alphabets are used for it. Situation of the remaining languages is not much different.

Tracing the history of languages in the region presently known as Pakistan, which first appeared on the world map in 1947, we observe that Sindhi was the main language and

medium of education in Sindh while in the remaining three provinces Urdu was mainly used as medium of education, although spoken native languages of people of these provinces were not Urdu. The result was that for all writing needs and correspondence Sindhi was used in Sindh and Urdu in rest of the three provinces. The three major languages of the region, ranked in order of written usage, are: Urdu, Sindhi and Pashto.

Alphabet of Sindhi was standardized as far back as 1850 and as per view of scholars and researchers of that time, of all the languages of Indian sub continent, Sindhi was richest in original literature. Urdu is the national language of Pakistan and also the most widely used language of the region.

Text Processing on Computers

To usefully discuss the implementation of languages on computers, we need to understand how computers handle text or written language. A paradox at first sight, but the fact is that a computer basically recognizes or knows only two digits: 0 and 1— based on presence or absence of current in a given bit. That is why computers are also known as binary machines. Using various combinations of these two digits (if there are 8 bits, total number of such combinations would be $2^8 = 256$), computer can create another set of numbers which it can then process or manipulate.

Since computer did not understand our language of characters, we had to find a way of communicating with it using its language of numbers. So we used this set of 256 numbers to talk to 8 bit computers, and this is how we did it: we created a table and assigned one of these numbers to each of the characters of the language. This arrangement of assigning number codes to characters came to be called code page or code plate of the language. Code page is an encoding scheme where every character is represented by a unique number so that the computer can understand it, process it.

In the beginning there was this 7 bit architecture which provided only 128 unique numbers or codes which could be assigned to the characters. This was sufficient for processing English. Later, as the need to implement other languages grew, the 8th bit was also used and stock of unique codes increased from 128 to 256. This facilitated handling of many languages including Arabic. For each language (apart from English, which continued to be the main language of the computers) a code page was designed and there came to be standardized hundreds of such code pages. Often there would be more than one code pages for a given language. And then there were languages, like those of Pakistan region, which did not even have a code page. Without indulging into a detailed analysis, suffices it to say that limited market and relaxed enforcement of copyright laws in the region were mainly responsible for unwillingness of the developers to commit resources for development of products for languages of the region. If there are no software products, no standards are needed; hence no need for code pages!

Unicode

The computer industry has been one of the fastest growing industries and the developments in the field have been phenomenal. The computing power has become abundant as well as affordable. This increase in computing power, more than anything else, ultimately made computing possible even in little known languages. As computer architecture evolved into 16 bits, the limitation of 256 unique codes was transcended in an explosive manner and 65,536 (that is what 2^{16} simplifies to) unique code points became available. This opened up new possibilities for multilingual and global computing. At the same time, growing use of internet and popularity of World Wide Web were also giving a new impetus and outlook to the industry. Need for platforms and products which could facilitate communications and networking at the global level started gaining currency.

In late 1980's work began on the development of a universal encoding standard based on 16-bit architecture that could address the needs of all the languages of world. After several years of informal cooperation, Unicode Consortium (a non-profit organization) was formed in 1991 to develop and promote such encoding scheme. The scheme came to be called the Unicode Standard (Unicode is acronym for Universal, Uniform and Unique Coding scheme). As per their own statement, the Consortium brought "together software industry corporations and researchers at the leading edge of standardizing international character encoding. The outcome of this cooperation is the Unicode Standard, which provides the foundation for internationalization and localization of software."ⁱⁱⁱ While each of the earlier and local standards, based on 256 code points as these were, dealt with one or two languages, the Unicode Standard, taking full advantage of 16-bit addressing system of newer processors, catered to the needs of all known written scripts and languages of the world. Thus Unicode, as an International encoding standard, has already started replacing these older encoding schemes. At the same time, by providing a universal code page, it provided a unique opportunity for little known languages to be represented on the computer platform.

Earlier, the various encoding systems inevitably conflicted with each other. Often there was more than one encoding systems for a given language. It was not unusual to find more than one encoding systems assign same number or code point to different characters, or vice versa assign different code points to the same character. To add confusion to the complexity, different encoding systems of the same language differed as to the character sets, implementation logic and rules, etc.

In this era of international computer networking and communications, there was this increasing need to have a universal referencing standard which could be used for transfer and exchange of data across the borders, computer platforms and applications. Computer stations were no more islands unto themselves. Now most of these machines were most of the time connected with other machines and networks all around the globe.

The Unicode Standard provided the common language needed for the global communications. Emergence and implementation of The Unicode Standard, besides making implementation of little known languages possible, also provided a robust foundation for internationalizing general computing, internet communications and

commerce. It was for these reasons that World Wide Web Consortium and Internet Engineering Task Force also adopted The Unicode Standard.

The Unicode Standard has been adopted by such industry leaders as Apple, HP, IBM, Microsoft, Oracle, Sun, Sybase and Unisys. Unicode is required by modern standards such as XML, Java, JavaScript, LDAP, CORBA 3.0, WML, etc., and is the official way to implement ISO/IEC 10646. The Standard is supported in many operating systems and by all modern browsers.

Unicode, a key standard for internationalization, is becoming more relevant and essential as web services and e-business become globalized. Until Unicode came along, there was little possibility of languages like those of this region benefiting from this new technology.

Current state of computing in languages of Pakistan

Having discussed in substantial detail the background, emergence and future promise of Unicode, let us now revert and inquire whether Unicode also makes computing possible in the languages that are understood by the masses living in region of Pakistan and its neighbors.

Before we do this, let us briefly examine the computing presently available for Pakistani languages. For the last almost 15 years, personal computers are being used for Urdu, Sindhi and Pashto. Mainly these are used for desktop publishing needs. From flyers to newspapers, almost every thing is typeset on the computers using different software packages. But how do the experts assess the present state of computing? Here are some observations in this regard:

Different applications have been developed by individuals and vendors since then, desktop publishing leading the scene for Urdu Software. As all these packages were developed without any underlying computing standard, each has its own character set and code page. Therefore data exchange between them is not possible. They even have their own keyboard settings, therefore making it difficult for a typist to switch from one application to another.^{iv}

Commonly it is understood that Urdu desktop publishing is Urdu computing. These programs are word processors not the programming tools which can be used to create other applications. We do not find any programming tools for Urdu as yet.^v

We have not even been able to make a dictionary in Urdu. How far is Urdu implemented as office language? It is not question of official support (availability of funds) alone, it also matters how far we as a nation patronize the effort. Whether it is development of applications or programming tools, it requires a great deal of financial investment. Now, if there is no market for the products, why would any one wish to invest and ultimately suffer substantial losses?^{vi}

These comments on the present state of computing pretty much tell the story. This is state of affairs in respect of Urdu, the most widely used and major language of the region. Sindhi and Pashto are in no way better off. The problems faced in the way of bringing computer technology to common user can be summed up as under:

1. There has been an utter lack of standardization resulting in mutual incompatibility issues and lack of transfer or exchange of data.

2. Most of the available software is for desktop publishing needs, which is used by commercial concerns for production of published works.
3. The languages of the region have not been implemented in offices and schools.
4. Due to limited market and piracy of software, the developers have not been willing to commit resources for developing software for languages of the area.
5. There has been practically nothing available which could be used for creating exchangeable documents, web pages, database management applications, sending and receiving emails, internet chatting etc. in the regional languages.

Having examined the nature of problems in the way of implementation of the regional languages on computers, and having also considered multilingual computing potential of The Unicode Standard, we believe that an excellent opportunity exists to achieve the objective of bringing computer technology to masses. We believe that the Unicode Standard, as implemented in MS Windows, the most widely used operating system for personal computers, decidedly offers the best and most practical solution for implementing the Arabic script languages of this region. This has the potential of bringing the technology within reach of common man. Now we can examine in more detail the basis for this view.

Advantages and Potential of Unicode for Pakistani Languages

Sometimes, to understand what a thing is, one has to at the same time understand what it is not. Unicode is not an operating system, a computer program or even a font creation system. It simply is an encoding arrangement of letters, characters and marks of all the written languages of the world, which provides a standardized reference framework for use in electronic text processing on computers. The Unicode Standard does explain, lay down and in some cases standardize the rules relating to the processing or shaping of world scripts, but it does not specify as to how an operating system or an application should apply these rules to correctly display or print the text. Correct rendering of text is handled by the operating systems and applications in their own way, though at the same time conforming to the basic requirements of the Unicode Standard.

Previously, to be truly multilingual, a computer had to learn and implement as many encoding arrangements as there were code pages – and even these did not cover all the languages of the world, in fact these did not cover a single language from the Pakistan region. Now, as the Unicode Standard was adopted, by learning just one encoding arrangement, the computer has become a truly multilingual tool.

Incorporating Unicode into client-server or multi-tiered applications and websites offers significant cost savings over the use of legacy character sets. Unicode enables a single software product or a single website to be targeted across multiple platforms, languages and countries without re-engineering. This has very far reaching implications. As stated elsewhere also, earlier it was not financially feasible to undertake development of software products for many languages mainly because there was not enough market to justify such investment. It was just another paradox: users of these languages, usually belonging to developing countries, needed software at affordable price; while the

developers, with their high fixed costs per package (fewer packages sold in limited market), had no option but to put a prohibitively high price tag.

Benefits of Unicode include:

- A standardized encoding system which can be used to represent text in any language. Documents of this type are portable across systems.
- The resultant ability to exchange text provides necessary foundation for technical and commercial developments such as web services and multilingual application integration.
- Increased support and availability of software for languages which, because of their limited market size, could not otherwise have been able to solicit such support.
- Easier localization of software and operating systems, which can truly globalize the IT and bring its benefits to all people and languages.

The Unicode Standard has brought an end to the era of inconsistencies, incompatibilities, redundancies and wasteful duplicated effort, by introducing a universal and consistent encoding scheme that holds a promise of globalizing the computer technology and making it available to languages and people across the globe, beyond the politico-linguistic barriers.

Though many sectors stand to benefit from the Unicode Standard, its importance for the languages using complex scripts like Arabic is indeed unique. Since the computer technology was, so to say, born and raised in the West, English became like mother tongue to it. The languages sharing similar script (i.e. Roman) also found easier implementation. But when it came to languages based on complex script like Arabic, it was a different story.

Needless to say, the software industry, like all other industries, is also driven by the economic factors of demand and supply. If there would be substantial demand for applications for a language, the industry would willingly employ its R&D resources to find solutions and meet the demand. That is how the demand for Arabic software in the oil rich Middle Eastern countries caused Arabic to be implemented on the computers.

Of all the languages using Arabic script, Arabic is not only the most sophisticated, the most ancient and the most widely used, but it also has the fewest letters/characters. As such, support for Arabic did not *ipso facto* result in support for the other Arabic script languages. This situation has continued more or less almost till today. But now, hopefully, things are changing. It has taken quite some time for the computing facility in these languages to emerge, evolve and mature.

With the development of internet and World Wide Web, the need for a common reference ground increased. But the mutual incompatibilities continued to be a major impediment in the way of effective communication and across the board implementation of these languages. The Unicode Standard came as a life giving breeze, especially for the languages with limited resources which would otherwise never have seen the light of the “computer” day. Had it not been for Unicode, many languages would never have support. This is because there is not large enough a market for these languages for the commercial software houses to financially justify development of software for these languages. And

even if software could have been developed, these would have been stand alone packages which could not have led to universal implementation on platforms such as World Wide Web.

Once the Unicode Standard is implemented in an operating system, it is much easier for the applications to incorporate support for various languages. Implementation of Unicode on Windows platform, and the resultant multilingual support provided in Microsoft's Office Suite exemplifies such benefits. Urdu has been major language of the region and as mentioned earlier there have been quite a few software packages around for it. But again, all these packages were standalone applications basically addressing word processing and desktop publishing needs. While Sindhi, though second most widely written language of the country, has not even been that lucky. Since 1988, computer has been used for desktop publishing needs, but only by hacking Arabic fonts for its use. There has been not a single software package for Sindhi. In 2000 first proper Sindhi font was developed conforming to the Unicode Standard and OpenType specifications. On being pointed out that MS Windows did not handle Sindhi properly, Microsoft started examining and modifying Uniscribe (the system library that implements Unicode and handles multilingual processing) so as to cater to the needs of the language. A prototype keyboard was also developed for Sindhi. Within a few months it became possible to use all the applications of MS Office suite for Sindhi, create Sindhi documents, web pages, send and receive emails in Sindhi and even chat in Sindhi. In pre-Unicode days, years of hard work and immense resources would have been needed to attain such an objective. Parkari is a little known language, spoken by about 250,000 people in Tharparkar district of Sindh. It uses Sindhi character set and three extra characters which were not in Unicode. Users of the language pointed this out and submitted examples of usage of these characters to the Unicode Technical Committee. The proposal has been accepted and very soon they too can use the whole range of MS Office products and such other products as conform to Unicode and offer multilingual support. This kind of opportunity did not exist at all for languages of this region until the Unicode Standard was implemented.

Criticism of Unicode

One may ask if Unicode offers such an opportunity then why we don't just make most of it. Yes, in fact this is high time that we did ask such question and search for some sound reasons for not doing that. While it is not easy to find any reason, good or bad, there has been some criticism of Unicode. Since this article basically deals with issues relating to the languages of Pakistan region, we would discuss only such criticisms as relate to these languages. Also, we must not forget the very objective that we started with: to find best means of implementing these languages on computer. In this regard, the only criticism that we came across related to implementation of Urdu.

A recent issue of National Language Authority of Pakistan's periodical Akhbar-e-Urdu (Special Urdu Software Issue) contained an account of proceedings of meetings of Urdu Computer Standardization Committees and some other articles which contained criticism of the Unicode Standard, explaining therein the grounds for questioning the suitability of

the Standard for Urdu and the justification for developing a separate and unique code page for Urdu. Some of the excerpts from these are translated and reproduced below.

Use of present Arabic Unicode is in no way suitable for Urdu, primarily because Unicode uses Naskh script and Urdu Naskh has never been acceptable. For Urdu we will need Unicode based on Nastaleeq.^{vii}

We need to get 65,000 Urdu characters registered [on Unicode]^{viii}

Some members [of the Urdu Standardization Committee] are of the opinion that Naskh should be used as the script of the computer... but it was pointed out that Naskh was never accepted as such for the computer (i.e. Unicode and internet) the standard script should be Nastaleeq and only Nastaleeq.^{ix}

The Unicode, which makes Urdu and other right-to-left languages mere sub-languages of Arabic, needs to be discarded. Instead we should develop our own Urdu Unicode so that Malay, Moroccan and other Central Asian languages could also use this Urdu Unicode and thereby avail all the facilities.^x

On international scene, most software companies now use International Code (Unicode). International Code (Unicode) presents Arabic Naskh typeface in its current repertoire. The National Language Authority is also pushing the proposal to get a place in the International Standard (Unicode) for Urdu Nastaleeq so that on international and technical levels, Urdu also is included in the list of international languages.^{xi}

Though naïve and without any serious substance, these criticisms reveal serious misconceptions about the Unicode Standard in particular and implementation of languages on computers in general. Coming as these are from serious and responsible quarters, these need to be addressed and clarified before we can move on to other more substantial issues.

Issues raised by these criticisms are:

- Unicode is Arabic
- Unicode is Naskh based
- Nastaleeq is a distinct script (رسم الخط)
- Nastaleeq alone is acceptable for Urdu
- Unicode Standard does not support Nastaleeq script
- Unicode makes Urdu sublanguage of Arabic

In its arrangement of characters, Unicode is script based and not language based. This makes sense too. Many languages share the same script and same characters. Instead of grouping the characters repetitively for numerous languages, these are all grouped together in the sub-range of the relevant script. Thus all the characters of languages based on Arabic script are grouped in the Arabic sub-range. Unicode is as much Arabic as it is Roman. It is like English speaking people turning down Unicode because it is Roman based or that it shows English as sublanguage of Roman.

Unicode is simply a character encoding system. It has nothing to do with how these characters finally get displayed on the computer screen. Thus it cannot be Naskh based or for that matter Nastaleeq based. Naskh and Nastaleeq are type style issues relating to

rendering which is handled by operating system and applications and not by encoding scheme.

There also exists this confusion as to script (رسم الخط) and a writing style (خط یا کتابت). Naskh and Nastaleeq are not two distinct scripts (not even subscripts, for that matter), like Arabic, Roman or Devanagari are. Instead these are just two of the many different writing styles (others being Kufi, Diwani, Jilli, Maghribi, Taleeq, etc.) of the same Arabic script.

That Nastaleeq and not Naskh should be the writing style used for computers is also based on this misconceived “Nastaleeq or Naskh” notion – which in turn is an unfortunate legacy of Urdu word processing packages which supported one style or the other. So far as Unicode is concerned, for example word Pakistan would always comprise of characters Pay, Alef, Kaf, Seen, Tay, Alef and Noon. And that is what is encoded by Unicode, or for that matter, that is what will be encoded by UZT 1.01 too. Now how does one encoding system (Unicode) become Naskh based, while the other (UZT) Nastaleeq based? Since the encoding system has little to do with the issue, at the best this criticism is misdirected. And before this article ends we shall show that based on the Unicode Standard and the rendering technology used in MS Windows, it is possible to use Naskh as well as Nastaleeq on the computers.

A simple example should settle the issue. If you ask a calligrapher to write word Pakistan in Nastaleeq and then in Naskh what will he do? He will first write the word in Nastaleeq and then using same characters, employing same rules of joining the characters and same Arabic script, he will write it in Naskh style. That is exactly what happens under the Unicode Standard implementation. It gives the system or application a text file containing characters which make up the word Pakistan. These are just Arabic script letters which form word Pakistan. There is no writing style involved at this stage. The system/application, like the calligrapher, renders those characters into a form that can be displayed or printed. The form can be Nastaleeq or Naskh. Unicode only deals with characters; it is the system or application which renders those letters into a form which we finally see.

Apart from the misconceived and misdirected criticisms discussed above, the Unicode Standard has also been criticized on following grounds:

- That Urdu alphabet has not been encoded in its natural order and Unicode does not specify collating sequence.^{xii}
- That Unicode does not fully represent the Urdu character set.^{xiii}
- Its use of double-byte encoding system increases the document size.

First two criticisms relate to Urdu and these are discussed at some length in the sections that follow. Here we shall briefly deal with the last criticism and then move on to the other two points.

This raises the issue of storage and transmission speeds. Larger the size of file, the more space it requires and longer it takes to transmit these. With the hard drive sizes running into gigabytes, 1.4 megabyte floppy being replaced by inexpensive 550 megabyte CDRWs, we hardly find occasion or reason to complain about the size or cost of storage media. The internet speeds are increasing too. If this criticism were to be extended, a case

could even be made for using the applications as were used 10 or 15 years ago just because these are fraction of the size of applications used today. But all said and done, there are Unicode transformation formats (UTFs), which can be used to represent the Unicode text in streams of bytes, thereby satisfactorily addressing this issue of document size.

Sorting order and related issues

It is correct that Urdu alphabet has not been encoded in its natural order in the Unicode Standard. But then neither does the Standard profess to address the sorting issues of the languages nor does it do so in respect of any other language. By ‘sorting’ we mean arranging words in an alphabetic order which in turn is based on some agreed upon standard. Such an arrangement is normally used in storing information (lists, dictionaries, tables of contents, etc.) to facilitate later retrieval, either electronically or manually.

Unicode has, for very good reasons, left the collating sequence be handled at software level which, properly speaking, is the place to address these. The sorting and collating issues can be addressed at the operating system and/or application level without any technical, schematic or logical problems. This approach also does away with the need to modify the code page each and every time some issue relating to collation is raised or settled.

Let us now see how the characters of various languages can be encoded in their natural order within a universal encoding scheme like Unicode. There can be two possible approaches:

1. All characters of all languages sharing the script (Arabic script in our case) be arranged in such a manner as to satisfy needs of all languages.
2. Group the characters in sub-ranges by language and not by script and then, within the sub-range, arrange these as per natural order of the language.

First of all, asking for any such changes in the Unicode Standard shows ignorance of its professed policy regarding character encoding stability. This policy requires that once encoded, a character cannot be moved, remove or renamed. Nevertheless, let us see if the two approaches are otherwise feasible.

As different languages sort same characters in different order, it is not possible to come up with a universal encoding order which would also agree with sorting order of all languages using Arabic script. Let us just consider how two languages, Urdu and Sindhi in this case, belonging to the same region, differ in their collating approaches. In the table that follows we have listed 4 characters that these two languages share, and then shown the relative collating sequence (other characters have been left out to simplify the comparison):

Sindhi ب ت ث پ

Urdu ب پ ت ث

As can be seen in Urdu پ precedes ت and ث while in Sindhi پ follows ت and ث. This shows that it is not possible to handle the issue in respect of scores of languages sharing Arabic script. A few years ago, when the code page for Urdu was being standardized, at one stage the standardization committees considered a code page as would also support Balochi, Punjabi, Pashto and Sindhi. Then this idea was shelved observing:

...each language has its own identity, character set and collating sequence, so each should be considered independently.^{xiv}

If collating sequence of just 5 languages of the same region cannot be addressed in one code page, how does one expect accommodating all the languages of the world in one code page? This shows that it is neither possible nor feasible to use the first approach.

The second approach requires that there be as many sub-ranges as there are languages to be supported. There being thousands of languages, the feasibility as well as functionality of such approach is very questionable. How many problems would such approach solve and how many more would it create in its wake? The second approach too is neither practical nor efficient.

While discussing the demand that Unicode address the issue and handle the sorting for Urdu, let us also look at the experience of the developers of UZT (the Urdu code page), and see how well they succeeded in doing so. This team of experts of computer science and linguistics admitted:

Sorting is a complex issue in Urdu because it is achieved through the characters *and* aerab [i.e. diacritics]... To enable correct sorting of words using computers, both levels of sort must be effectively implemented. It was not possible to achieve both levels of sorting directly through the code page... Level two sort[ing] must be achieved through software.^{xv}

If sorting could not be implemented in a code page made just for one language, how could one expect it to be implemented in a code page which deals with all the languages of the world? Moreover, if some level of sorting needs in any case to be done at the software level, then what could be the justification or logic for splitting the task and having it done in parts at two places? Anyhow, we may now ask: Is it possible at all for an encoding scheme to satisfactorily resolve this issue, especially in case of complex scripts like Arabic? The answer is a resounding no.

We observed that the Standardization Committee for Urdu found it impossible to make a common code page for Urdu and four other languages of the region. Later it even admitted that it was not possible to completely address the sorting issue at code page level even for Urdu. This is because Arabic is a complex script and the languages do vary as to the character sets and collating sequence. These very arguments, *inter alia*, provide the reasons for Unicode not attempting to handle this issue at the code page level.

Finally, let us face that any attempt to address this sorting issue at code page level must inevitably presuppose an existence of standard character sets and collating sequences for the languages to be supported. Do such standards exist for languages of this region?

Let us begin with Urdu which, apart from being the National and Official language of Pakistan, is also the most widely used and spoken Arabic script language in the world after Arabic itself. While discussing the development of Urdu Computing Standards, the leading experts in the field conceded that:

Different authors have quoted different number of characters in Urdu alphabet (e.g. even the elementary books for children do not agree on the same alphabet. Kifayat (1993), Siraj (1999), PTBB [Punjab Text Book Board] (2000), BUQ (1999) and KUQ(1999) have 36, 51, 53, 47, and 37 characters respectively...) ... As no general agreement was available, the [standardization] committees agreed to consider the alphabet used by the National Language Authority (NLA), which contains 57 characters...^{xvi}

As late as year 2000, there was no agreement as to the number of characters in Urdu alphabet. This is a very basic issue which always needs to be settled before one can move on to consider collating sequence – whether at code page level or at software level. It is strange that the Unicode Standard is criticized for failing to encode Urdu alphabet in its natural order when that very order has been lacking.

Here is the grave ramification for languages of the region. Presently scores of languages using Arabic script have been included in the Unicode and of these only a few have this issue sorted out, viz. Arabic, Persian, Sindhi, and may be Pashto. If sorting could somehow be handled at code page level, then before these languages could be included in the code page, there had to be an agreed upon or formally standardized character set and sorting order for the languages. And if there is no such standard, as there is not for most of the languages of the region, then these languages would remain from being included in the code page. How would this in any way help the languages of the region? This would have meant exclusion of all those languages from the Unicode Standard till such time that sorting related matters have all been resolved and standardized.

How Microsoft implements linguistically sensitive sorting

Microsoft Windows implements linguistically sensitive sorting in the NLS (National Language Support) layer of the operating system. This means that it is possible to have different sorting orders for different languages which otherwise are using the same Unicode sub-range for their character sets. Presently Windows ships with collation support for Arabic. Collation support for Urdu and Sindhi is being considered and may be available in future updates of Windows Xp. It also becomes incumbent upon the users of a language to raise these issues with Microsoft and get support for their language. Official bodies, such as Language Authorities set up by the governments in this region, can play an effective role to guide and assist the Windows International development team in their effort to research and implement linguistic collation for the language. Thus, it is possible to use the operating system support for languages of Pakistan and get the sorting done in proper order.

But how do we get the sorting done till then? Implementation of sorting at the application level is always possible. Not only that, where necessary, the applications can override the system or provide alternate methods of sorting too. Operating systems and application developers can also evaluate the possibility of user determined sorting. As of writing this, it may not be possible to do sorting for Urdu and Sindhi in Microsoft applications like Word, Excel or Access, but it is always possible to develop database applications which use the Unicode encodings as reference points to implement the linguistically appropriate sorting order. While bemoaning the temporary lack of this feature for these languages in MS Office suite programs, one must not overlook the fact that use of all other features of these applications became available only through implementation of Unicode.

Representation of Pakistani languages in the Unicode

Now we can discuss the criticism that the Unicode Standard does not fully represent the Urdu character set. Earlier we discussed the disagreements as to what constitutes Urdu character set. Such disagreements exist in respect of other languages also where formal standardizations remain to be implemented.

The Unicode Standard is committed to an encoding arrangement which caters to the needs of all written languages of the world. This does not mean that all the characters and letters of all the scripts and languages are actually represented in the Unicode. The developers of the Standard seem to have undertaken considerable research to fulfill their commitment to comprehensively represent all the languages of the world, but their success, or lack of it, also depends upon the extent of participation by users of the languages. Many of these languages lack not only the computing standards but also the standards and agreement as to the content and arrangement of alphabet. The responsibility to standardize the character set and then see to it that it is adequately represented in the Unicode Standard, falls squarely on the shoulders of the users, proponents and custodians of the language. Unicode is an evolving standard which also relies on feedback from the users of the languages to fulfill its basic commitment.

The Unicode Standard version 1.0.0 came out in 1991. Its major update, version 3.0.0, was released in September 1999. At that time in Pakistan, the Urdu text books taught in the elementary schools, did not even agree as to the number of letters of Urdu alphabet. This number varied from 36 to 51. Failure of a Standard to properly represent character set is always judged with reference to a well established standard at home. Suffice to point out here that by that time there was hardly any such standard at home.

By this time, based on the characters included in the Unicode and the multilingual support available in MS Windows, it had already become possible to use computers for Urdu. At any rate, a study was done that concluded:

An exercise was done to identify the Urdu characters in Arabic block and draw up a table of comparison. The result is given in Table 1. After the exercise was completed it was found that 25 characters do not have a representation in Unicode.^{xvii}

Following is the list of these missing characters:^{xviii}

#	Missing Character	Remarks
1	Decimal Sign	Dropped
2	Colon Sign	Resubmitted
3	Hard Space	Dropped
4	Hamza e Izafat	Dropped
5	Kasra e Izafat	Dropped
6	Alef Below	Accepted
7	Pesh Above	Dropped
8	Inverted Pesh	Accepted
9	Zare Below	Dropped
10	Small Tah	Resubmitted
11	Sakoon	Resubmitted
12	Reverse Sakoon	Accepted
13	No Diacritic Sign	Dropped
14	Ligature Bismillah	Resubmitted
15	Ligature Alahe as Salam	Accepted as mark
16	Ligature Radiallah	Accepted as mark
17	Ligature Rehmatullah	Accepted as mark
18	Takhallus Sign	Accepted as mark
19	Misra Sign	Resubmitted
20	Footnote Sign	Accepted
21	Safah Sign	Resubmitted
22	Number Sign	Accepted
23	Sanah Sign	Accepted
24	Long Madd	Dropped
25	End of Section	Dropped

It may be observed that none of the letters of the alphabets has been found missing. Mainly diacritical marks, punctuation marks, special signs, etc. have been missing. Of these 10 were accepted for inclusion in Unicode, while nine were dropped after discussing and conferring the matters with technical experts. This left six that have been made part of another proposal (along with 10 other characters – the 10 digits) which has recently been sent to the technical committee of Unicode.

This shows that the initial view that 25 characters were missing was not accurate. Since then at least nine characters went out of the list and 10 new made their way in. This simply shows a possibility that the analysis regarding less than perfect representation of Urdu characters in Unicode could itself also be less than absolute.

Since the Unicode Standard is committed to defining codes for characters of all the major languages of the world, there is no reason why Urdu characters would be left out. It is only matter of proper research, presentation, coordination and understanding.

However, depending again on the nature of these missing characters, a deficiency in the encoding system should not prevent users from taking advantage of it. For decades, ASCII served as code page for English despite the fact that it did not contain codes for such frequently used typographic marks as true single quotes, double quotes, en dash, em dash etc. Urdu character set present in the Unicode is indeed much more comprehensive than ASCII character set was for English. The smart approach would be to make use of it and at the same time to get it improved.

The following table lists the alphabets of Pashto, Urdu and Sindhi, the three major languages of Pakistan, and shows the corresponding Unicode code points for each of the characters. This shows that the alphabets of these languages are fully represented in the Unicode.

Pashto	Sindhi	Urdu
ټ	ټ	ټ
0627	0627	0627
ټ		ټ
0622		0622
ب	ب	ب
0628	0628	0628
	ب	
	067B	

	پ	پھ
	0680	0628 + 06BE
پ	پ	پ
067E	067E	067E
	ق	پھ
	06A6	067E + 06BE
ت	ت	ت
062A	062A	062A
	ث	تھ
	067F	062A + 06BE
ٹ	ٹ	ٹ
067C	067D	0679
	ن	ٹھ
	067A	0679 + 06BE
ث	ث	ث
062B	062B	062B
ج	ج	ج
062C	062C	062C
خ		
0681		
	جھ	جھ
	062C + 06BE	062C + 06BE

چ	چ	چ
0686	0686	0686
خ	چ	چ
0685	0687	0686 + 06BE
ح	ح	ح
062D	062D	062D
خ	خ	خ
062E	062E	062E
د	د	د
062F	062F	062F
	ذ	ده
	068C	062F + 06BE
ڊ	ڊ	ڊ
0689	068A	0688
ذ	ذ	ذ
0630	0630	0630
	ڍ	ڍ
	068D	0688 + 06BE
ر	ر	ر
0631	0631	0631
		ره
		0631 + 06BE

پ
0693

ژ
0699

ڑ
0691

ڑھ
0691 + 06BE

ز
0632

ز
0632

ز
0632

ژ
0698

ژ
0698

ک
0696

س
0633

س
0633

س
0633

ش
0634

ش
0634

ش
0634

بن
069A

ص
0635

ص
0635

ص
0635

ض
0636

ض
0636

ض
0636

ط
0637

ط
0637

ط
0637

ظ	ظ	ظ
0638	0638	0638
ع	ع	ع
0639	0639	0639
غ	غ	غ
063A	063A	0634A
ف	ف	ف
0641	0641	0641
ق	ق	ق
0642	0642	0642
ک	ک	ک
06A9	06AA	06A9
کی	ک	کھ
06AB	06A9	06A9 + 06BE
	گی	گی
	06AF	06AF
	گھ	گھ
	06AF + 06BE	06AF + 06BE
ل	ل	ل
0644	0644	0644
		لھ
		0644 + 06BE

م
0645

م
0645

م
0645

مه
0645 + 06BE

ن
06BA

نه
06BA + 06BE

ن
0646

ن
0646

ن
0646

ني
06BC

نه
06BA + 06BE

و
0648

و
0646

و
0646

ه
0647

وه
0646 + 06BE

ي
064A

ه
06BE

ه
06BE

ي
06D0

ة
0629

ى
06CD

ء
0621

ء
0621

ئ	ي	ى
0626	064A	0649
ى		
0649		
ے		ے
06D2		06D2

Better Choice

Having discussed the criticisms against Unicode, an observation is in order as to the choice of better solution for computerization of languages of the region. Not many choices are there. Urdu is the only language for which at least a code page has been recently standardized, so apparently that might be seen as an alternate choice of implementing it on computers.

Implied in all these criticisms against Unicode is a hint, if not assertion, that relative to Unicode the UZT (Urdu code page) may be better suited to meet the needs of Urdu language. It may be so, but we must not overlook the fact that an encoding system, however good, needs an efficient rendering engine to complement it and make it usable. In case of Unicode, a tested and developed rendering engine is available complete with an operating system and a set of applications which implement it. As of writing these words and to our knowledge, no such rendering engine, OS or set of applications are available to complement UZT and thereby make it available, either now or in immediately foreseeable future, for computerizing Urdu, let alone other languages of the region which it does not even profess to support.

Without discussing the relative merits and demerits of the two encoding schemes, if nothing else its immediate usability makes Unicode a better choice – that is, if the immediate computerization of the languages is the objective. It is obvious that it will be a while before UZT can acquire the functionality and usability to make itself as useful for Urdu as Unicode and Windows environment are at this time. As far as the rest of the languages of the region are concerned, no solution other than Unicode is in sight.

The Character/Glyph Model and Rendering on Windows OS

As explained in earlier sections, Unicode is a character based encoding system. In this section we will explain how these characters get rendered into readable text which is finally displayed on the computer screen or printed on the paper. At the very outset, we would like to explain two terms, viz. *character* and *glyph*, as used in this article. The distinction between the two is very important to understand before we can grasp the concept and process of rendering.

Character is defined in Unicode as the smallest component of written language that has semantic value, while *glyph* is defined as the shape that a character or characters can take when they are rendered or displayed. Natural language consists of characters, while a digital font contains glyphs. A Unicode text file *always* contains reference to characters, never to glyphs.

The Unicode standard requires that the Unicode text strings be input and stored in a simple logical sequence. Thus if we wish to write “Pakistan” we would press in succession the keys corresponding to the characters which make up Pakistan and the text file would store the Unicode code points of these characters. The code in the backing store of the text file would look like this (underneath we show the corresponding characters for illustration purpose only and these would not be in the file):

0050	0061	006B	0069	0073	0074	0061	006E
P	a	k	I	s	t	a	n

What happens when this file or text needs to be displayed? The glyphs corresponding to each of the successive code points are displayed in the default left-to-right order without any reordering, regrouping or substitution whatsoever. There is usually one to one relationship between characters and glyphs in case of Roman script. This is what makes handling of Roman script straight forward and simple.

If we write in Urdu a word, such as پاکستان, again we would press in succession the keys corresponding to the characters that make up the word. The text file would store the Unicode code points of these characters in logical order, as in the earlier case. The code in the backing store of the text file would look like this (the characters underneath are only for illustrations purpose):

067E	0627	06A9	0633	062A	0627	0646
پ	ا	ک	س	ت	ا	ن

Now when an application or an operating system has to display this file or text on screen, unless it has some rendering engine, it will be displayed just as it is. But if the Arabic script text is displayed as it is, it will appear as follows:

ناتسکاپ

This would be far from acceptable as neither the ordering is correct nor the shapes of the characters are as per rules of the script. But that is how it would appear if there is no rendering engine to process the text. That is why the Unicode Standard also presumes existence of some kind of rendering mechanism, operating either at system level or at application level, which can layout the complex script text in proper order (i.e. right-to-

left in this case) and also substitute proper glyphs that represent the initial, medial, final or isolated form of the character as its context requires.

In Microsoft Windows environment, the rendering is accomplished by integrated use of three system tools or technologies, viz.:

- OpenType font format
- Windows' Unicode Script Processor (Uniscribe)
- Open Type Library Services (OTLS)

Using simple terminology, we will try to explain how these three elements work together to render text properly and in accordance with the prevalent rules and norms of the script/language. This would also expose us to the potential of this technology and how it can be used for Pakistani languages. We must however keep in mind that all these three sophisticated tools/technologies, which have evolved into what these today are after years of extensive testing, make use of and are based on the Unicode Standard.

To understand how Windows rendering tools layout and display the text, we will follow progress of a simple word in Urdu and see how it makes way from input to rendering. We start with characters and finally see the word made up of appropriate glyphs.

Let us now follow the journey of word پاکستان from input to rendering. Here are the characters in the backing string of this simple word, as input by user:

Character	Code Point
پ	067E
ا	0627
ک	06A9
س	0633
ت	062A
ا	0627
ن	0646

The back store of the file would contain the code points as shown in the right column in the right-to-left order as under:

067E 0627 06A9 0633 062A 0627 0646

The rules relating to the layout and processing of Arabic script characters are explained in the Unicode Standard.^{xix} Unicode also standardizes the properties of the characters. Uniscribe contains the needed information and rules regarding shaping and rendering of the text of various scripts. There are basically two kinds of operations which Uniscribe carries out to render the text properly, viz.

1. Correct ordering of text.
2. Shaping:
 - a. Substitution
 - b. Positioning

For example, when Uniscribe encounters the code point 067E, it right away knows that it is dealing with a script that is written right-to-left. Accordingly it reorders the character for display in right direction. At this point, if it were to display the text without carrying out shaping operations, it might appear like this:

پاکستان

It would be because shaping rules have not yet been applied and normally in a font the glyphs representing the isolated forms are mapped to the code points of the characters.

Next, Uniscribe analyzes the character for contextual shapes and, using OTLS, substitutes and positions the correct glyph obtained from the OpenType font. As Uniscribe encounters each character, it analyzes its properties, the properties of its neighboring characters, and determines which of the four (initial, media, final or isolated) forms is needed and using OTLS retrieves the appropriate glyph from OT font and displays it. For example in case of Pay, Uniscribe would know that its initial form is needed and the OT font is asked to provide the required glyph. This way, all the glyphs needed to represent the characters are obtained and instead of the glyphs representing the isolated forms, the glyphs representing the appropriate are substituted and used to form the word. We first tabulate the position and then show the word finally rendered on screen:

Text Entered → Rendered

Character	Code Point	Form Needed	Glyph Used
پ	067E	Initial	پ
ا	0627	Final	ا
ک	06A9	Initial	ک
س	0633	Medial	س
ت	062A	Medial	ت

ا	0627	Final	ا
ن	0646	Isolated	ن

This table lays out the rendering process in a simplistic way. Column on the left shows the character key pressed by the user. Next column shows the corresponding Unicode code point which is put in the file and later used by system to render the text. Third column describes the appropriate form of the character needed in the given context as determined by Uniscribe, the rendering engine. Finally, using OTLS, Uniscribe gets the glyphs which represent the required forms of these characters and displays these as under:

پاکستان

Previously the code pages usually contained code points for all the shapes of a character and the fonts would map the glyphs in one to one relationship with all the shapes encoded in the code page. When the OS or application needed to render the character it would be simple matter of retrieving and displaying the corresponding glyphs. Or, if the code page did not encode all the forms, it would then specify the mapping of the font, so that proper glyphs could be used.

But Unicode and OpenType architecture have changed this. Unicode not only does not bind the fonts as to the location or mapping of the glyphs, it does not bind it as to the number of glyphs either. The actual number of glyphs needed to display the text properly would depend on the orthographic style supported by the font. A simple Urdu font may have glyphs numbering less than hundred and these may be enough to adequately display the text. While a Nastaleeq font may need thousands of glyphs.

Under the OpenType architecture, all of the decisions as to number of glyphs, types of ligatures, substitution and positioning rules, are left to the discretion of the font designer. Of course a font is expected to conform to the standards and specifications, but beyond those minimum requirements, it is left to the discretion of the font designer. For example, an Urdu font must contain glyphs which could provide various shapes needed as per rules of script and language.

An OpenType font contains tables which include all the information that is needed for the interaction with the operating system tools and/or applications. These tables incorporate all the substitution and positioning rules, mapping structure of the font etc. Uniscribe uses OTLS to interact with an OpenType font and apply these features relating to substitution and positioning of the glyphs accordingly.

OpenType font technology has literally opened up the doors to creative and flexible typography without in any way compromising the standards or norms. That which was once impossible to achieve employing the relatively simpler, but at the same time constrictive, digital font architecture as typified by the original TrueType format, can now be easily done using the flexible and powerful OpenType architecture. Once the potential of OpenType architecture is understood, then it is easier to visualize how it can be employed to implement complex scripts like Arabic and handle intricate writing styles

like Nastaleeq. Special relevance of this technology with reference to Pakistani languages will also then become apparent.

Now we can discuss why and how is it that not having a fixed standard as to mapping of glyphs within a font makes the architecture more suitable for complex script languages. This gives the font designer flexibility of design and choice which he/she can take to the limits using the OpenType features and the willingness of application developers to support these features. The designer has the choice to use as many or as few features as are needed for the particular font.

To explain this we will now compare two fonts. Times New Roman, so far as its Arabic script part is concerned, is a simple design functional font which makes use of just those features as must be used to render text properly. As against that, Arabic Typesetting is a sophisticated Naskh font which makes extensive use of the OT features to achieve a very high degree of typographic excellence. By examining the differences between these two fonts we shall learn about the flexibility and the potential of OpenType font architecture.

For letter Bay, the Times New Roman has just two glyphs, which are used for four different positions: initial, media, final and isolated. Examine the following sample:



The glyph at far left is used for initial form as well as for the medial form. In earlier original architecture, we could use one shape but we would have to create two glyphs, albeit identical: one for the initial and one for the medial forms. But in an OpenType font we may have just one glyph, in one address. In the OT tables we just say that this same glyph is to be used for these two positions. Similar is the case with its isolated and final position. Same glyph is used for both the positions:



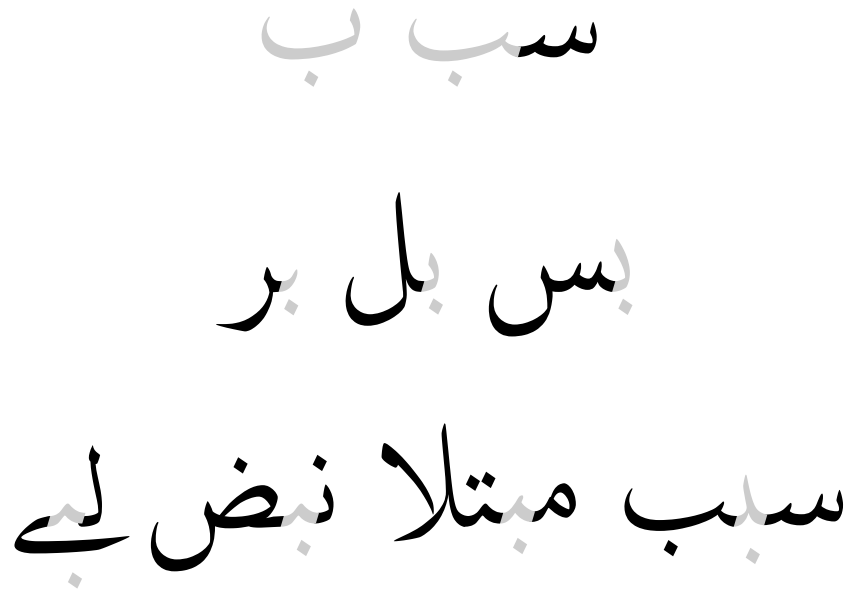
The glyph at the left is used for isolated as well as for final positions of Bay.

Before moving on to Arabic Typesetting font let us examine another font which has four different glyphs, each for the initial, media, final and isolated forms.

Isolated	Final	Medial	Initial
ب	ب	ب	ب

When Uniscribe asks for a glyph for a particular position, the font examines its tables and supplies the appropriate glyph. Thus, it is an internal design and arrangement issue for the font. The number of glyphs it has for a particular form will depend on the font design.

Let us now finally look at the examples from Arabic typesetting font which has in all nine different glyphs for Bay. There is one each for the isolated and the final positions, three for initial position and as many as four for medial position. Before discussing which glyph is used when and why, let us look at the examples showing these shapes.

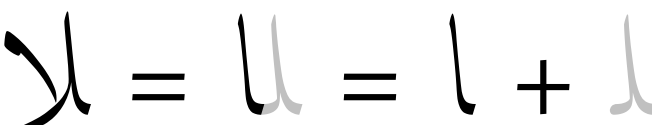


As can be seen, there are three varieties of initial form of Bay and four different glyphs for its medial form. When Uniscribe needs the initial form, it just asks for it, so to say. If the font does not have any rules to supplement the basic rules of character/glyph substitution, it simply provides the only glyph which its table links to the character as its initial form. But if a sophisticated font has further rules, these may define that if initial Bay precedes certain glyph, or a set of glyphs, then a particular glyph should be used. Similarly if the medial form is used in a given context, then a particular glyph may be

used. The initial Bay before Seen is not the same as one before Ray. Same goes for the medial shapes. Detailed rules define which glyph would be used in which context. This shows the potential of OpenType. All these levels of simplification or complexities are not bound by the parameters of the system, instead these are left for the font designers to put in place and system faithfully implements these rules as and when it finds them.

The examples which we have just examined use the substitution feature of OpenType font architecture. There is another very powerful implementation of this feature and that is in respect of ligatures. Ligature is simply a glyph representing an alternate rendering of a group of glyphs. In Arabic script there is at least one general case where the rules of writing demand use of ligature, otherwise the formation would be incorrect. Whenever a Lam is followed by an Alef, the normal rules of joining are not used, instead a ligature is used. This is what happens:

0644 0627 

0644 0627 

Whenever a glyph representing initial or medial forms of Lam is followed by Alef, the one would expect that the glyph representing final form of Alef would be used. But the ligature rules in place in the font would substitute the two glyphs with the glyph representing the appropriate ligature form. On the left we show the code points in the backing store. Note that despite all these substitutions taking place, just one glyph being used, the backing store would always contain reference to two characters: Lam and Alef. Whatever glyph or glyphs may be used, Lam and Alef remain Lam and Alef.

This is known as required ligature as per rules of scribing and it must always be used instead of normal joining forms. But if we look at examples of calligraphy, we would find extensive use of ligatures, which may be called optional or discretionary ligatures. These ligatures may or may not be used depending on the writing style.

Earlier, the font structure and the code page used to lay down strict rules in this regard and would provide what ligatures may be used. The font design had to conform to these rules and was accordingly restricted by it. Like it or not, use it or not, glyphs for the ligatures had to be put in place. The Unicode Standard, following a strict character approach, does not deal with ligatures at all. This simply means that there are no restricting rules imposed by the encoding scheme. On the other hand, the OpenType architecture also has left this issue totally open for the font designer. The font designer may choose to have no ligatures in a font or he/she may choose to have as many ligatures

as the design of the font demands. Creativity of the designer and not the technology defines the limits. So far as the system and applications are concerned, these also do not deal with this issue directly. When a font is used, all the rules of substitution and positioning, as incorporated in the font, get automatically executed. Let us now look at some more examples of such ligatures as used in Arabic Typesetting font (incidentally the Arabic Typesetting font, which is scheduled to be bundled with Office .NET, includes complete Arabic character set covering all the languages such as Urdu, Sindhi, Pashto, etc. that use Arabic script):

Without Ligatures	Ligatures
الله	الله
محمد	محمد
نبي	نبي
يشرب	يشرب
سرائي	سرائي
عجم	عجم
لحم	لحم
راسخ	راسخ
في	في
حجت	حجت

Above we have given just a few examples of ligatures just to show what is possible to achieve using OpenType font architecture. The font from which above examples have been taken actually has more than 800 ligatures in its repertoire. A good thing is that,

depending of course on the kind of implementation provided in the applications, the user is not bound to use the ligatures if there are instances warranting use of normal joining forms. In the above table, same font is used but in the column on the left, MS Publisher's option to disable Optional Ligatures has been used. Thus one may use ligature for a word, and then turn it off for the next. Then there are also discretionary ligatures or forms, where the user, as he enters the text, is given choice to use one of the alternate forms. This advanced typographic feature has been included in some applications, such as Adobe's InDesign. Corresponding version with Arabic support has not been released as yet, hence it is not possible to say what level of support will be included for Arabic script.

Substitution is one feature which is used for the flexible typographic design. Positioning is another feature which can be used to fine tune appearance of Arabic script text. Positioning features are extensively used for accurate positioning of diacritic marks (the *a'rab*). Following example shows how the positioning of diacritics can enhance the text:



The font on the first of these two lines has taken care of positioning of the diacritic marks, whereas the font on the bottom line has yet to implement the positioning of marks. These features let the font designer set the position of marks relative to glyphs or other marks precisely. These features that can now be implemented on personal computers and in very simple applications like WordPad and NotePad, were earlier the domain of high end typesetting applications using proprietary codepages. This is the kind of flexibility and the level of control which OpenType font architecture gives to the font designer.

Positioning feature can also be used for kerning, that is relative positioning of two characters. In the following example, we show how kerning can be used to enhance the text and give it more natural look.



Figure 1 - Left side does not have kerning applied. Right side has kerning applied

Unlike the traditional kerning, in which only horizontal position could be manipulated, now it is possible to adjust the positioning in any direction. This feature can also be used to greatly enhance visual appearance of text and implementing writing styles like Nastaleeq.

Now finally, we can look at the following example. There is no such word, but we have formed it just to exemplify the capability of OpenType structure.

بیسبیب

Using cursive positioning feature it is even possible to adjust the vertical connective positioning of the glyphs. It is no more necessary to have all the glyphs sitting on the baseline. The font designer can define the exit point of the glyphs and have them connect at levels other than baseline, as is done in above case.

OpenType format has plenty of features which can be used in the design of the font to achieve the typographic excellence which once was handled by the sophisticated typesetting applications. New features, to meet the needs of font designers, can always be added to the already rich repertoire of features presently offered. If we have to write a word such as نستعلیق we can employ either of the following styles:

نستعلیق

نستعلیق

نستعلیق

In all these cases it is the same set of character code points (that is code points for ن س ن ق ی ل ع) which gets rendered using same code page, same rendering engine but only different font each time.

All these nice features, as someone has very aptly put it, are there not merely to prettify the text but to articulate it^{xx}. All the time, the digital typography aims at emulating the excellence and visual eloquence achieved by human genius of master calligraphers. We may still be far from that goal, but the present set of tools has certainly brought us closer to it.

Conclusion

Computer users in Pakistan are poised at a junction and they have to make choice and make it before it is too late. The facility to use mainstream computing for all computing needs, from word processing to publishing on the web, from chatting on the internet to web commerce and database management, and even creating developmental tools in regional languages, is readily available. Computer technology is for the first time available in local languages and it can be used in businesses, offices, schools and homes alike. Either one can take advantage of it, and then use the available resources and energies to go from this point forward; or one may choose to employ these resources to reinvent the wheel, duplicate what others have already done and indulge in redundant activities. All this will, at the best, get us where we are right now. At some date in future we will commence the same journey which we can commence today, only then we will be a few years late. And by that time the technology will have already taken another leap forward, we will again be struggling to catch up (if we can) and all the while the masses will be deprived of the use of this technology.

About the Author

Abdul-Majid Bhurgri belongs to Larkana, a town in Sindh. In the 1980s, Abdul-Majid was the first to implement a solution for Sindhi on Apple Computers. He was also a part of the civil service of Pakistan. In 1972 Abdul-Majid was selected for Income Tax Service on the basis of the Competitive Superior Civil Services exam. Later, he resigned from the service. Abdul-Majid currently lives near Seattle in the USA.

ⁱ Afzal, M. and Hussain, S. 2001 *Urdu Computing Standards: Development of Urdu Zabta Takhti (UZT) 1.01*

ⁱⁱ Ethnologue *Languages of Pakistan* http://www.ethnologue.com/show_country.asp?name=Pakistan

ⁱⁱⁱ See the Unicode Consortium website at <http://unicode.org>

^{iv} Afzal, M. and Hussain, S. 2001 *Urdu Computing Standards: Development of Urdu Zabta Takhti (UZT) 1.01*

^v From interview of developers of first Urdu Database for Libraries, published in Jan/Feb Issue of National Language Authority of Pakistan's periodical *Akhbar-e-Urdu* Jan/Feb 2002 Issue

^{vi} *ibid*

^{vii} p 99 *Akhbar-e-Urdu*, Jan/Feb 2002

^{viii} p 101 *ibid*

^{ix} p 101 *ibid*

^x p 101 *ibid*

^{xi} p 104 *ibid*

^{xii} Zia, Dr. Khaver *Towards Unicode Standard for Urdu*

^{xiii} *ibid*

^{xiv} Proceedings of Standardization Committee for Urdu as quoted by Afzal, M. and Hussain, S. 2001 *Urdu Computing Standards: Development of Urdu Zabta Takhti (UZT) 1.01*

^{xv} Afzal, M. and Hussain, S. 2001 *Urdu Computing Standards: Urdu Zabta Takhti (UZT) 1.01*

^{xvi} Afzal, M. and Hussain, S. 2001 *Urdu Computing Standards: Development of Urdu Zabta Takhti (UZT) 1.01*

^{xvii} *ibid*

^{xviii} *ibid*

^{xix} *The Unicode Standard Version 3.0.0* § 8.2, pp 189-198

^{xx} Hudson, John, *Windows Glyph Processing* Microsoft Typography Website