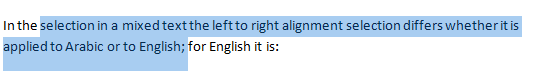
The space taken by a word differs whether it is applied in the middle of the word or at the end or at the beginning.

Sometimes a word shrinks when we add suffix to it (spacing problem) how to react regarding the text. Same if we have a word and we remove some characters and it get longer!

In the selection in a mixed text the left to right alignment selection differs whether it is applied to Arabic or to English; for English it is:

في اللغة العربية لكي نحدد مقطعاً وتعليمه تكون العملية معكوسة فيبدأ من اليمين إلى الشمال كما يظهر في هذا المثل فنلاحظ أن طريقة تحديد المقطع تختلف





**Line and Word Breaks**

Line-breaking and word-wrapping algorithms are important to text parsing as well as to text display. Western languages typically follow patterns that break lines on hyphenation rules or word boundaries and that break words based on white space (spaces, tabs, end-of-line, punctuation, and so on.).

However, the rules for Asian DBCS languages are quite different from the rules for Western languages. For example, unlike most Western written languages, Chinese, Japanese, Korean, and Thai do not necessarily distinguish one word from the next word by using a space. The Thai language does not even use punctuation.

For these languages, world-ready software applications cannot conveniently base line breaks and word-wrapping algorithms on a space character or on standard hyphenation rules. They must follow different guidelines.

For example, the kinsoku rule determines Japanese line breaking — you can break lines between any two characters with several exceptions:

* A line of text cannot end with any leading characters — such as opening quotation marks, opening parentheses, and currency signs — that should not be separated from succeeding characters.
* A line of text cannot begin with any following characters — such as closing quotation marks, closing parentheses, and punctuation marks — that you should not separate from preceding characters.
* Certain overflow characters (punctuation characters) can extend beyond the right margin for horizontal text or below the bottom margin for vertical text.

**Mirroring Awareness**

For Right-To-Left (RTL) languages, not only does the text alignment and text reading order go from right to left, but also the UI layout should follow this natural direction. Of course, this layout change would only apply to localized RTL languages.

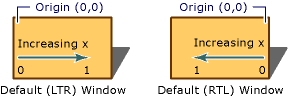
**Note**The .NET Framework does not support mirroring.

Arabic and Hebrew Windows 98 introduced the mirroring technology to resolve the issues with flipping. Windows 2000 uses this same technology. It gives a perfect RTL look and feel to the UI. For Windows 98, this technology is only available on localized Arabic and Hebrew operating systems. However, on Windows 2000 and later, all versions of the operating system are mirroring aware making it possible for you to easily create a mirrored application.

To avoid confusion around coordinates, try to replace the concept of left/right, with the concept of near/far. Mirroring is in fact nothing else than a coordinate transformation:

* Origin (0,0) is in the upper RIGHT corner of a window
* X scale factor = -1 (i.e., values of X increase from right to left)

The following figure illustrates the coordinate transformation from LTR to RTL:



To minimize the amount of re-write needed for applications to support mirroring, system components, such as "GDI" and "User," have been modified to turn mirroring on and off with almost no additional code changes except for a few considerations regarding owner-drawn controls and bitmaps.

**Fonts**

Windows has the ability to select an appropriate font to display a particular script. Windows accomplishes this by using a new face name called MS Shell Dlg. MS Shell Dlg is a mapping mechanism that makes it possible for Windows to support cultures/locales that have characters that are not contained in code page 1252. It is not a font, but is instead a face name for a nonexistent font. The MS Shell Dlg face name maps to the default shell font associated with the current culture/locale. For example, in U.S. English Windows 98 this maps to MS Sans Serif. However, in Greek Windows 98, this maps to MS Sans Serif Greek. In U.S. English Windows 2000, it maps to Tahoma. However, MS Shell Dlg does not work on East Asian versions of Windows 9x. For more information, see [Localization and the Shell Font](http://msdn.microsoft.com/en-us/library/ms776266%28v=VS.71%29.aspx).

However, application developers often overlook fonts when creating world-ready applications. Here are two issues that you must watch when dealing with fonts:

* Hard-Coded Font Names — With the use of Unicode, we now deal with thousands of different characters instead of hundreds. Most fonts do not cover all of the Unicode character set. Thus if you hard code a font name that displays English characters and not Japanese, all of your localized Japanese text will display incorrectly. Another reason not to hardcode font names is that the font you want may not be on the system that is displaying your text.
* Hard-Coded Font Sizes — Some scripts are more complex than others. They need more pixels to be displayed properly. For example, most English characters can be displayed on a 5x7 grid, but Japanese characters need at least a 16x16 grid to be clearly seen. Whereas Chinese needs a 24x24 grid, Thai only needs 8 pixels for width but at least 22 pixels for height. Thus, it is easy to understand that some characters may not be legible at a small font size.

The best way to treat font names and sizes is to consider them as another localizable resource. Using MS Shell Dlg solves the problem of running your (any language) application on (any language) Windows NT/Windows 2000. Setting your font as a localizable resource solves the problem of making it possible for your localizer to change the font for the localized UI.

For Arabic the upsizing and downsizing of a sentence is not the same as in English; the spacing is not scaled accordingly neither the width between the characters is saved..

**Complex Script Awareness**

The special processing required by a complex script can involve one or more of the following characteristics: character reordering; contextual shaping; display of combining characters and diacritics; specialized word break and justification rules; cursor positioning; filtering out illegal character combinations. Scripts considered complex are: Arabic, Hebrew, Thai, Vietnamese, and Indic family.

It is important to respect these following points:

* When displaying typed text, do not output characters one at a time.
* To allocate character/glyph buffers, do not assume one character equals one glyph.
* To measure line lengths, do not sum cached character widths.

**Bidirectional Awareness**

Bidirectional (Bidi) is the term used to describe text that has scripts that flow both left-to-right (LTR) and right-to-left (RTL). Text that consists of a mixture of English and Arabic is a good example.

There are several issues you must keep in mind when making sure your application is Bidi-aware.

* Internal Data Storage — As mentioned above, Bidi text has LTR and RTL flowing scripts. Although both scripts flow differently, both are stored in the same order from first character to the last character. The best way to envision this is to think of the data stored from the top of a buffer to the bottom.
* Display Stream — Most Latin-based languages are displayed one character at a time. Bidi-text's different properties of character position, which prescribe script flow and how Arabic ligatures change their shape depending on the preceding and following character, have changed this display formula. Now, it is best to save the currently displayed line in a buffer and then output the whole buffer every time you modify or add a character in the line.
* Line Length — Because of the ligature changes mentioned in the bullet above, it is not a good practice to sum cached character lengths to calculate the length of a line.

Language Issues

Language issues are the result of differences in how languages around the world differ in display, alphabets, grammar, and syntactical rules.

<http://msdn.microsoft.com/en-us/library/aa292134%28VS.71%29.aspx>

right to left, bi-directionality, more text less width, less text more width, wrapping to a new line...

Then move to discuss selection and rendering Arabic selections. Inversion of selection areas and flikkering on drap... How to define a good rectangle, what are the alternatives...

Discuss wrap lines and why they are good versus keeping the whole text in lines.

Discuss the issue of copy/paste where we have to copy an Arabic (or bidirectional string) into another string. How does that affect rendering and wrapping.

## 1.2 Character Shaping and text direction algorithms

The graphic form (glyph) shown in the ISO 8859-6 character chart is not the identity of that character (like in Latin 1 for example). The same Arabic character may correspond to up to four different glyph types. The glyph type of a character depends on the position of the character within a word. The possible glyph types are: the Beginning shape (a character that begins a word), the Middle shape (a character that is surrounded by other Arabic characters), the End shape (a character that is located at the end of a word) and the Standalone shape (a character that is surrounded by whites paces).

Some characters can be linked to another character on either side, (each character has four possible glyphs), some characters can be linked only on their right side (and have only two possible glyphs) and some characters cannot be linked on either side (and have only one possible glyph).Also, in some character sequences, the formation of ligatures is obligatory. These ligatures associate one specific character form to the joining of two Arabic characters. They are necessary for well-rendered Arabic text. In addition to this, Arabic text is written from right to left, and mixed Arabic/Latin strings include text in both directions which is presented on the same line. In fact, text is stored in sequential order in the backing store. Logical or backing store order corresponds to the order in which text is typed. The conversion from backing store format to the readable one represents one of the major problems in the processing of Arabic script. In order to be the most useful, powerful and especially transparent regarding applications, this conversion must be handled by low-level text rendering routines.

## 1.3 Character fonts

The Arabic character font also has another particular need. Arabic is always written in "cursive" or "handwritten" form, where characters are linked together as if they where written by hand. The linking rules are well defined, but the font needs to be adapted to this style, and the display device must be able to join all characters designed in order to avoid blank columns between characters. The nicest solution is to use proportional width fonts which are very important for rendering Arabic script. Some devices, such as alphanumeric terminals, cannot handle this kind of font, and use fixed width fonts instead; the result is less enjoyable but remains readable.

## 1.4 Global Screen direction and mirror effect

Due to the Right to Left writing direction of the Arabic language, the common way to read a document is to start from the top right-hand corner. This is also the case for an application screen or a printed document. This characteristic of the Arabic language is also a problem for standard applications. These applications are designed for Latin based character sets which have built their screens starting from the top left-hand corner position. Also, several little details, such as the menu cascading in a GUI (*Graphical User Interface*) application, need to be right to left oriented. These characteristics are very difficult to localize if they are not included in the original design of the software.

## 1.5 Numerals and Hindi Digit shapes

Numerals are mainly handled in the same way as Latin languages. Numbers are read from left to right with the highest order digit on the left side. However, there are two possibilities for numeral shapes. In North African countries, the digit's glyphs are the "Arabic digits" (i.e. the same as for Latin use). In Middle Eastern countries, the digits used are the "Hindi shapes." The display of one of these two possible digit representations must be user configurable.

## 1.6 Arabic vowels and collating sequences

The Arabic vowels (named "Tashkil") have a specific status in Arabic text. In fact, in common use, these characters are simply not used and Arabic text is written only with consonants. Any possible synonym confusion is cleared according the context. However, in some cases (e.g. official or legal documents), vowels may be added to the text. These vowels look like Latin accents that are displayed above or below a consonant letter. The problem here is that from a collating sequence or a pattern search point of view, a word with vowels and the same word without vowel must have the same intrinsic value. Also, text input with vowels must be displayed or printed with or without vowels.

## 1.7 Neutral characters

Normally, Arabic characters are always written from right to left, and Latin characters are always written from left to right. For some technical reasons and in order to be able to display text correctly in right to left mode some application screens or forms are initially built in a left to right direction. Some characters must be able to take the global writing direction despite their own direction value. It is necessary then to define a set of neutral characters which are able to use the global writing direction when being written.

## 1.8 Dual keyboard management

In most cases, European language keyboards have one specific keyboard layout, including all needed Latin letters. Since the Arabic alphabet is different from the Latin character set and because a user must always be able to input Latin and Arabic characters from the keyboard (ISO 8859-6 includes both ASCII and Arabic characters), a dual keyboard management system is needed. The keyboard management system must allow the user to switch from one language to the other using a single keystroke. This is also the case for languages that use Cyrillic, Greek, or Thai character sets.

In order to be coherent, the solution must include either an engraved keyboard with both ASCII and Arabic letters on each key, or to be more flexible, a set of keyboard stickers to be installed by the user on the existing keyboard. Sometimes the second solution is not acceptable to users and it is necessary to supply an engraved keyboard.

.

## 1.9 Optical Character Recognition

Because Arabic is a connecting letter language, it is quite difficult to use the same method and algorithms for Optical Character Recognition as for Latin languages. The main problem is the ability to extract a single letter from a word.

# 2. Arabic Standards and Culture

## 2.1 Handwriting

One of the most difficult points of the Arabic writing culture is the ability to reproduce cursive handwriting aspect of the text on a computer screen or within an application. Initially, with "character based" interfaces such as typewriters, dump terminals and dot matrix printers, results where acceptable, but were not appreciated by all users. This kind of interface always uses fixed width fonts and, even when "context analysis" and "automatic shape determination" are implemented, the final rendering (on a screen or on a printed document) is not high quality.

Since the appearance on the market of graphical interfaces (such as Microsoft Windows on PCs and X Window on UNIX platforms) and the ability to use proportional width fonts and WYSIWYG (*What You See Is What You Get*) systems, the rendering of Arabic text on a screen or on a PostScript printer more closely reflects user expectations.

## 2.2 Codeset uses

Some Arabic standards, like the keyboard layout, are conflicting and are not always clear. The ASMO, standards organization for Arab countries, has determined a specific layout, but user habits have developed from the use of specific manufacturer products different standards. For example, regarding the ligature "Lam-Alef," this ligature (visual shape coming from the juxtaposition of two ISO letters "Lam" (L) and "Alef" (A)) appears on an IBM keyboard standard. To be compliant, Microsoft has also implemented this same solution. Since this character does not exist "as is" in the ISO codeset, the ASMO layout does not include this key, and most users find this unacceptable. In fact, this problem is due to market competition rules that give a de-facto standard to the first solutions. The localization of an application must also take into account these kinds of problems.

## 2.3 Local differences

Arabic writing is the same for all Arab countries. However, the speaking of Arabic may be different and therefore different country's must be implemented. This is the case for month names, which are different in North Africa, around the Nile Valley area and in the Middle Eastern countries.

## 2.4 No abbreviations

Abbreviations and acronyms (like IBM for International Business Machine, NY for New York or PM for time indication) do not exist in Arabic. We always need to specify a complete word. This characteristic needs to be taken into account when translating messages and labels of user interfaces into Arabic. The length of messages will grow. In some cases, the real visual length of labels and message should be greatly reduced using a proportional width font. One of the characteristics of Arabic cursive writing is that the beginning and the middle shapes are more narrow than final or isolated shapes. The result of a normal string is very condensed.

## 2.5 Justification of text

The general behavior of some applications is not directly applicable to Arabic language usage. This is typically the case for word processing and their justification features. The standard algorithm to justify text in Latin is to add spaces between words on the line in order to achieve both left and right alignment. The method used for text justifying in Arabic is to stretch the last letter of a word in the line. This stretching is called *keshide,* and to be fully compliant with the Arabic culture, a word processing must implement it.

<http://www.langbox.com/staff/arastub.html>

Diacritics

• Zero-width characters

كتب

أبي

أب

كلّمت البارحة بعد الظهر أبا فؤاد فقال لي دع الأمر لي

كلّمت البارحة بعد الظهر أب فؤاد فقال لي دع الأمر لي

أين

كتبنا

التنسيق العام

التنسيق العام

أهلاً وسهلاً

أهلاً وسهلاً

أ

# 3. Rendering Issues

Text rendering is the process of assigning characters to glyphs. For some languages, rendering is simple, while others pose a big challenge in displaying text, because of special diacritics, direction, ligatures, and contextual forms. Rendering of characters of such languages needs a complex processing.

The Arabic language has 28 letters which consist of consonants and a few long vowels. The so-called "short" or "light" vowels are placed above and below characters, and their representation is a huge challenge. Generally, short vowels are used in a very few cases, e.g. in the holy Coran, where the exact pronunciation is very important. However, this problem remains unresolved for sophisticated information retrieval systems. Hence, the question arises, how to represent such vowels. To represent each vocalized character as single code is not efficient, because we will need a very large repertoire of codes. Additionally, it is sometimes desirable to display such text with or without vowels.

Hence, vocalization (or diacritics) in Arabic poses a big issue in character rendering and in retrieval as well. Other issues are that most operating systems do not support Arabic character code sets, keyboard mapping, date and time format, etc. Existing of some ligatures poses further problems. For instance, the LAM-ALEF ligature is mandatory. Moreover, Arabic is a cursive language, i.e. characters are linked together. Rules exist to define how to link characters, but the font still need to be adapted to this form.

An Arabic character might take on four different glyphs depending on its position in the word, i.e. at beginning, in the middle, surrounded and standalone. Some characters present only one or two glyphs. Another aspect is the bi-directional direction of writing. Arabic text is written from right to left, while Arabic digits and Latin characters are shaped from left to right.

Arabic characters are stored in logical order different from the visual order. The logical order corresponds to the reading/typing order. This means that the internal storing structure, which is based on ISO or other standards codes, does not correspond to the visual structure which depends on the contextual form of the Arabic language. The conversion from the logical into visual order is the main issue in rendering Arabic. The cut&paste and mouse selection can be not straightforward implemented. Extensive processing is obligatory.

In Arabic, there are often the problem that documents written in one system (e.g. Macintosh) can be not displayed on some browsers, because of the existence of a number of different code pages. That means, an Arabic glyph or letter has different character code among those various code pages. This is one of the biggest issue of developing Arabic application that can be used worldwide, such as an Arabic browser. Many publishers, like news agencies, still provide their information as images via WWW, because no real software can help them publishing and displaying their information resources correctly and properly. This solution leads to user frustration because of slow download and search inability.

From the article:

Towards Arabic Rendering Issues - MHTML Approach

H. Haddouti1, A. Maeda2, T. Sakaguchi3,   
S. Sugimoto3, K. Tabata3

2Nara Institute of Science and Technology, Japan

3University of Library and Information Science, Japan

1 FORWISS (Bavarian Research Center For Knowledge-based Systems)

Orleansstr. 34, 81667 Munich, Germany

[haddouti@forwiss.tu-muenchen.de](mailto:haddouti@forwiss.tu-muenchen.de)

One of the biggest concerns with Arabic localization is text direction. Arabic and Hebrew are considered “bi-directional” languages. Bi-directional languages, also referred to as “BiDi”, refer to text input and output that uses a mix of Latin (usually English or French) text along with the target language. Arabic and Hebrew are the primary BiDi languages, but they also include Farsi, Urdu, and other languages using right-to-left (RTL) script.

As big of a challenge as bi-directional text poses to the Western developer, other issues not previously encountered in localization loom large when internationalizing and localizing for Arabic. In fact, when we as localization professionals with over 30 years of experience began examining a few of these issues: contextual analysis, rendering and shaping, alternate numeric display, Hijri dates, character extenders for justified text, “logical vs. output” order, neutral characters, symmetric swapping; we realized that we needed to create and study an entirely new glossary with definitions for some of these terms before we could even get around to resolving said issues.

Companies that do not deal with the numerous challenges that language poses in a development environment on a regular basis face a daunting task in reaching Arabic-speaking markets.

Without any uncertainty, Arabic localization challenges require careful thought and resource planning when translating content or applications to this language for the first time. Fortunately, the features included in BiDi

Windows make development of a localized version of a standard Windows application relatively simple. In addition, simple recognition of Arabic conventions at the development stage can prevent many of these issues to become overwhelming challenges in subsequent localization phases.

**Arabic Language Features**

There are a number of obvious differences between English and Arabic, but the differences fall into five main categories:

1. Arabic **characters are laid out in RTL** (Right to Left) order, *with the exception of numbers*, which are laid out in LTR (Left to Right), likeEnglish. Text is right aligned on the page, and written from top to bottom.

2. Letters **change shape** depending on context. Each letter has up to four forms:

a. **Initial form**, which is the first letter in a word

b. **Final form**, when the letter is the last in a word

c. **Medial form**, when a letter is surrounded by other characters in

a word

d. **Isolated form**, when a character stands by itself (one letter

word)

3. Arabic can include **diacritics**. Such marks, placed above or below

letters, typically represent vowel sounds or other modifiers.

a. Diacritics are primarily used in children’s and religious text, but

are supported by BiDi Windows. (For this reason, diacritics are

not discussed further in this paper.)

b. Vertical kerning of diacritical marks is required for acceptable

output.

4. Arabic makes extensive use of **ligatures**, (the process whereby two

letters printed together are replaced by a single new character.)

English employs such ligatures as “ff” and “fi”.

5. **With Arabic, numbers may be represented by either** Hindi digits

or Arabic digits, depending upon the target region or country for the

Arabic content. Numbers are displayed LTR in all BiDi countries.

**Layout**

“Layout” involves differences in how text is input (Arabic and Latin) and

stored vs. how text is physically displayed. Display is mixed because Latin

characters and numerals display LTR, while Arabic words display RTL. Arabic enabled applications and/or systems always include a logical-to-physical transformation algorithm. All text input is in *logical* order—all output (e.g. screen display or hard copy) is in *physical* order.

The physical appearance of a bilingual string can differ enormously from how it is stored internally in an application (e.g. user input in software.) Noncontiguous cursor movement, text selection, and semantic re-ordering of sections of text must be taken into consideration for BiDi-enabled applications.

The intention of Unicode is to be able to display mixed-language text without carrying data about the text. Unicode can solve many layout/text display challenges, but there are situations in BiDi in which text display can be less than desirable. A good example is the hyphen.

In Arabic text, how should the logical string **494-9571** be displayed?

 If this is a telephone number, the entire number should be displayed as shown.

 If it is a subtraction, it should be displayed **9571-494**. (e.g. RTL display for “**494** minus **9571**”)

The computer cannot tell how to properly display this text without additional information from the user. To resolve such ambiguities, Unicode provides support for right-to-left marks and left-to-right marks. These non-printing characters tell the system how to interpret the direction of characters that follow.

**Contextual Analysis (Shaping)**

The Arabic alphabet has thirty-six alphabetics (no upper case), ten numeric symbols, and a few special alphanumeric characters. An Arabic code page usually consists of all these symbols combined with the English alphabet.

All Arabic alphabetic characters can have **up to four display representations** depending on their relative position in a word: *initial, final, medial,* or *isolated.* Its left and right neighbors determine an Arabic character’s shape. The algorithm determining which of the four shapes to use is called *contextual analysis*.

The Figure below shows the four forms of the Arabic characters ‘ain,’ ‘ba,’

‘qaf’ and ‘ha’ respectively, indicated in **red**.

**Initial Isolated Medial Final**

****

If your localized software application requires a significant amount of user input, contextual analysis is an important design concern. Examples of such software would be word processors, text editors, or software that requires extensive comments, notes and annotations to be input by the user. In regular translation, the four character forms are also a significant challenge for proofing content.

As indicated, shaping is the process of selecting the appropriate glyphs to

represent a set of codes. The Unicode term for this process is *rendering*, and it is also sometimes referred to as *glyphing*.

Another shaping issue is *symmetric swapping*. This is the process of **changing the direction of a neutral character** depending on language.

The characters concerned are “bracket” characters: < and >, [ and ], ( and ), and { and }. In Latin, we write the logical string 3 < 4 as 3 < 4, while in RTL, it is displayed as 4 > 3 (RTL display for “3 is less than 4”.) Because it is read RTL, the shape of the less than symbol ( < ) has changed.

**Ligatures**

As in English, certain combinations of two (even three) characters form one shape. (In English “ff” and “fl” are common examples.) Ligature selection is dependent not only on the characters themselves but also on the selected Arabic font. Some fonts do not use ligatures at all and others may have as many as 200 different ligatures defined. Only 63 ligatures found in the Traditional Arabic font are supported by Arabic Windows.

Traditional Arabic Ligatures supported in Arabic Windows

****

**Text justification**

A Latin paragraph is justified by inserting spaces between words. Arabic line segments are justified by *stretching the connectors* between certain letters inside words.

Stretching is accomplished by adding an extra character called a *kashida* which looks like a horizontal connecting line. Arabic calligraphic rules, whichfavor some characters to be extended over others, will influence where kashida is implemented. Kashida use is also a font property and it is incorrect to use kashida justification for certain Arabic fonts.

Arabic text incorrectly justified with space between words

****

Arabic text correctly justified with kashidas (extenders)

****