Bidirectional Wordprocessing With AbiWord

Dec 6, 2003

Tomas Frydrych <tomas@frydrych.uklinux.net>

Contents

Introduction

Some languages, such as English, are written from left to right, while other languages, such as Arabic, from right to left. AbiWord can handle both directions of text, as well as their combinations -- AbiWord is a bidirectional word processor.

Controlling Bidirectional Ordering

The bidirectional ordering of text in AbiWord is done automatically, closely following the Unicode Bidirectional Algorithm (UBA; see the Unicode Consortium website). The Unicode character set assigns each character certain directional properties which are then used by the UBA to order text. Thus, Hebrew or Arabic characters will automatically be treated as right-to-left, and English characters as left-to-right. There are some characters that are directionally ambiguous, and how they are treated by the UBA depends on what characters are found in their vicinity (this includes all white space and punctuation characters).

Sometimes it is desirable to have the characters ordered differently than the following the UBA. In AbiWord the user has at his or her disposal three basic mechanisms that allow him or her to fine-tune the results. These are *specifying dominant direction of text*, *overriding implicit directional properties of characters*, and *inserting direction markers*.

Dominant Direction of Text

The same sequence of characters with different directional properties will look differently if it is assumed to be a left-to-right text with right-to-left text embedded in it, or if it is understood to be a right-to-left text with left-to-right text embedded. In AbiWord we refer to the basic direction of

text as the *dominant direction* (in the the Unicode documentation it is known as the base embedding level). The dominant direction in AbiWord operates on four hierarchical levels: paragraph, section, document, and the program.

Paragraph-Level Dominant Direction

The paragraph-level dominant direction can be set either by the *Right-to-left* dominant check box in the Format->Paragraph dialogue, or from Format->Direction, or by using the equivalent button on the Extra toolbar. If the dominant direction is not set explicitly by the user, AbiWord will work it out from the rest of the dominant direction hierarchy, i.e., it will check if dominant direction is set explicitly for the section in which the paragraph is located, and if not, it will check the document level settings, finally resorting to program-level defaults.

Section-Level Dominant Direction

The section-level dominant direction is controlled by the *Use RTL order* check box of the Format->Columns dialogue. Apart from providing the default for any of the section paragraphs that do not have their dominant direction set explicitly, the section dominant direction controls how columns in multicolumn sections are ordered, it determines the dominant direction of text of footnotes inserted into the section, and the order of columns in tables. If section-level dominant direction is not set, AbiWord will derive it from the rest of the dominant direction hierarchy as described earlier.

Document-Level Dominant Direction

The document-level dominant direction is derived from the program-level dominant direction at the time when the document is created. So, if your program-level dominant direction is set to RTL, every new document will have its default direction set to RTL. At present there is no way to change document-level dominant direction in an existing document (this is going to change in future versions).

Program-Level Dominant Direction

This is the value to which AbiWord recourses if everything else fails. The program-level dominant direction is set in the preferences (Tools->Preferences->Language->Bidirectional options). The default preference value is set to LTR (if you build AbiWord yourself from the AbiWord sources, you can change the default preference value to RTL).

Explicit Direction Overrides

The visual order of characters that is automatically produced by the UBA might not always be what the user needs. AbiWord allows the user to specify explicitly that certain characters should be treated as left-to-right or right-to-left irrespective of their Unicode properties. This is done by selecting the characters in question and then applying the direction override from the Format->Direction menu or using the corresponding buttons on the Extra toolbar.

It is important to understand that the direction override is in fact a formatting property applied to the text. The consequence of this is that when you place the insertion point into or just after text with the override set, any new characters input will also have the override set. For example if you set the insertion point just pass text which has override set explicitly to LTR and then type a

Hebrew or Arabic character, it too will have the override set and will be treated as if it was LTR. To remove the override, you proceed in a manner analogous to setting it.

Direction Markers

Setting explicit direction override might sometimes not be the best way of changing the order of characters, particularly if the ordering is to be changed for one of the directionally ambiguous characters. The Unicode character set contains two special characters called direction markers: LRM (left to right direction marker) and RLM (right to left direction marker). The sole purpose of these characters is to allow small adjustments of the bidirectional order by affecting properties of ambiguous characters in their vicinity: when the text is reordered these markers behave as normal left-to-right and right-to-left characters, but when the text is displayed they are not shown (that is, unless you have Show Formatting Marks turned on). The LRM and RLM markers can be inserted using the Insert->Direction Markers menu, or by using keyboard shortcuts Alt+Ctrl+> and Alt+Ctrl+< respectively.

Examples of Using Direction Markers

The use of the markers is best shown on a couple of examples such as writing formulas and phone numbers.

As a first example, we will take the formula *log(x)*. If it is embedded into right-to-left text (represented here by capital letters ABCEFG), it will look like this:

GFE (log(x CBA

This is because the UBA will disambiguate the closing parenthesis of the formula to RTL (the algorithm does not know, and does not care, which is the opening parenthesis it matches; it takes an approach that will more often than not produce desired result, i.e., it assumes that a closing parenthesis on a direction boundary will have the properties of the characters that follow it). However, in our case the order we want is:

GFE log(x) CBA

This can be achieved by following the closing parenthesis with the LRM marker. The inclusion of the marker will make the parenthesis completely surrounded by LTR characters, and so it will behave as an LTR character.

The same result could, of course, be achieved by selecting the closing parenthesis and applying to it an explicit left-to-right override as described in the previous section. The main disadvantage of using an override in case like this has to do with the persistence of the explicit override described in the previous section. In contrast, the marker only affects the characters in its immediate vicinity, and only those that are directionally ambiguous. So if you insert RTL character just after the LRM marker, the new character will still behave as RTL, not LTR.

Another situation in which these markers come handy is with phone numbers. For instance, if the phone number 123 456 789 is embedded into a right-to-left text, it will look like this:

FED 789 456 123 CBA

This is because English digits are considered weak LTR characters: they will be ordered from left to right themselves, but any ambiguous characters embedded among them will derive their

direction from directionally strong characters that surround the whole number segment. In our case those are the ABC, DEF characters and so the spaces between the three groups of numbers behave as right-to-left characters. If, however, what the user wants is a phone number that looks like this:

FED 123 456 789 CBA

all that is requires is that the LRM character is inserted just before typing in the first digit; this will make the following numbers behave as strong LTR characters, and consequently the spaces too will behave as LTR characters.

Automatic Insertion of LRM and RLM Markers

In certain circumstances AbiWord is capable of inserting these direction markers automatically, based on the keyboard layout used (this is currently only supported under Windows). In order to use this feature you need to first of all make sure that the option to change language when changing keyboard layout is turned on (Tools->Preferences->Language), and that also the bidirectional option *auto-insert direction markers* is turned on (Tools->Preferences->Language). AbiWord will then follow all closing parenthesis (Unicode characters ')', ']', and '}') with a direction marker derived from the language applied to that character. For instance, if a ')' character is set as being written in Hebrew, it will be followed by a RLM marker but if it is set as being in English, it will be followed by a LRM marker. Similarly, it will precede all opening parenthesis ('(', '[', and '{' characters) with an appropriate direction marker.

This feature could easily be extended to other characters; if you would find it useful, file a request in our Bugzilla.

Mirroring Characters

Mirroring characters are characters the glyphs of which need to be mirrored when displayed in RTL context. An example of a mirroring character is the opening parenthesis, which looks (in LTR context but) in RTL context. When it comes to these characters the Unicode definition is strictly semantic, i.e., opening parenthesis has always the same numerical code, but when found in RTL context the application is expected to display in its place the mirror glyph of LTR opening parenthesis, which happens to be the glyph associated with LTR closing parenthesis. The effect of this is that if you display RTL text with parentheses that follow the Unicode rules in a plain text editor that is not Unicode-compliant, you will see '(' where you would expect ')' and vice versa.

AbiWord, as a Unicode-based application, complies with the Unicode rules for handling mirroring characters. The consequence of the above is that your keyboard has to generate semantically correct values for the mirroring characters. **On some Unix system this is not the case**, and the keyboard for languages such as Hebrew generates the code for closing parenthesis in place of the code for opening parenthesis and vice verse. If you are seeing ')' when you are expecting '(' and vice versa, you need to fix the keyboard definition file (how to do that is beyond scope of this document).

Glyph Shaping

Closely related to AbiWord's bidirectional capabilities is its ability to change visual appearance of certain glyphs depending on their context. This is essential for correct handling of the so-called mirroring characters described above, as well as for languages that use scripted alphabets, such as Arabic, in which each letter has different shapes depending whether it stands alone, or at the beginning, in the middle or at the end of a word. Alongside this type of glyph shaping, AbiWord can also replace a sequence of two glyphs with a special ligature glyph where needed.

At present AbiWord uses a proprietary shaping engine of fairly limited capabilities. We are currently working on getting adequate support for Arabic, and support for other languages that require shaping can be added on request. However, the built-in shaping engine can only handle languages for which the alternative glyph shapes have separate code points assigned to them in the Unicode character set (such as Arabic); some languages that were added to the Unicode character set relatively recently rely solely on advanced font technologies for shaping and these will not be supported in near future (e.g., Syriac).

When shaping and replacing ligatures, AbiWord always checks for the presence of the replacement glyph in the currently selected font. If the glyph is not available it will use the original character(s), providing they are available. If even the original characters are absent, AbiWord will first try to remap them to sensible values, but if event that fails, it will use the absent glyph character.

An important thing to understand about the glyph shaping is that the changes only take place in the visual plain (on screen or paper print out), but the characters that are contained in the document do not change in the process.

Glyph shaping is controlled from the Language tab of the Tools->Preference dialogue. There are two check boxes there: *Determine glyph shapes from context* and *Use glyph shaping for Hebrew*. The former of these is the master-switch that turns the glyph shaping engine on and off. When the second check box is checked, the shaping engine will shape also the five Hebrew letters that have a final form. **Please note that this is not intended to be used for writing modern Hebrew and Yiddish documents**, since in modern Hebrew and Yiddish the final forms are considered different characters (and because as I have explained above the shaping does not change the characters in a document, if nothing else, your spell-check will not work, and your files will not look right on other people's computers).