Antikythera Publications

DATABASE DESIGN NOTE SERIES

Relational Database Design
http://www.AntikytheraPubs.com
sweiss @ AntikytheraPubs.com

Exploring Bidirectional Text Entry and Editing
Multi-script Database Series #5

Prepared by: S. L. Weiss and F. Oberle

Like the subject of the previous Design Note in this series (Evaluating Fonts for Multi-Lingual Documents), entry and editing of bidirectional text is not specifically a database design issue. But, as noted in that paper, a typical user's perspective will often be that if you were the one providing the data, whatever issues arise must certainly be your responsibility.

Entering and editing multilingual text can be far more confusing than simply using a keyboard with a different alphabet and continuing to type. Before releasing a fully globalized schema to production or even initial user testing, database designers and application developers need to be familiar enough with some basic issues in order to respond to any user confusion that may arise. This also implies going beyond simple entry and editing in "data fields" to editing larger blocks of text such as might be found in reports, word-processing documents and so forth. This tutorial will get you started.

15 December 2016

See page 15 for information on other material from Antikythera Publications.

www.AntikytheraPubs.com

Database Design Note Series on Multi-Language/Multi-Script Databases

1. Exploring Alphabets
2. Exploring Complex Text Layout
3. Exploring UTF-8
4. Evaluating Fonts for use in Multi-Lingual Documents
5. **Evaluating Bidirectional Text Entry**
6. Evaluating Bidirectional Text Handling Behavior in Applications (planned)

Database Design Note Series – Exploring Bidirectional Text Entry and Editing

Contemporary word processors and other applications usually support paragraph-level mixing of bidirectional scripts. If right-to-left Arabic and left-to-right Cyrillic scripts are each *restricted to their own data fields or paragraphs* in a document, there are usually no difficulties with text entry – other than being able to easily type the "foreign" characters (see the sidebar). Mixing bi-directional scripts *within a single line*[1], however, can be confusing with some applications. It can be even more confusing, and often unsuccessful, if one or more segments includes bracket pairs – [{ (*text*) }] for example – or similar mirrored symbols such as "quotation" 'marks.'

The intent of this paper is to familiarize the reader with the entry and editing techniques used for bidirectional text.[2] Hebrew is used as the example of a right-to-left Script. Arabic would have been equally valid (as would N'ko, Syriac, Thaana/Thâna or Urdu), but Arabic's heavy use of Contextual Character Forms and Shaping[3], and the fact that many readers may never have heard of the others, made Hebrew seem the most appropriate choice.

An Assumption …

… is that the reader will use an IME (input method editor) to switch keyboard layouts when typing multi-lingual (or multi-script) text, since the examples are predicated on a user being able to just "type" the characters used. Most examples are still valid if letters are entered by other means, but such methods are tedious. In most operating systems, IMEs are easy to activate and deactivate, so their use is encouraged in order to follow these examples painlessly.

English keyboard equivalents are given for each "foreign" character, so ***no knowledge of Thai, Hebrew or Greek is required to try the examples*** using an IME. With I-Bus (the IME used while writing this), the Hebrew keyboard used was "Hebrew (lyx)" since "Hebrew (Biblical, Tiro)" doesn't have parentheses – used to illustrate several difficulties often encountered with bidirectional text entry.

Following the Character Entry Sequences in the Examples:

Each keystroke to be made is indicated by the character that would be typed on an English keyboard when using an Input Method Editor. An a , for instance, means to press the "A" key to type a small (lower case) "a" character. Similarly, Shift+a means to type a capital "A" using either "shift" key. With a Greek keyboard mapping (λ) selected in the IME, pressing u results in a "θ" (theta); pressing Shift+w results in a "Σ" (capital sigma).

Regardless of how the IME is invoked, it is not counted as a keystroke in these examples even though one or more "shortcut" keystrokes may be used to activate it.

To illustrate how to duplicate the examples, we'll begin with two Scripts having the same directionality and type: "This is English and ภาษาไทย is Thai." Begin with En selected.

	Legend:
ก	Switch IME to "Thai (TIS-820.2538)" Keyboard
א	Switch IME to "Hebrew (lyx)" Keyboard
λ	Switch IME to "Greek (polytonic)" Keyboard
En	Switch IME to "English (US)" Keyboard
spc	Abbreviation for the Space Key; otherwise it's space.
Shift	Either Shift Key; e.g. Shift+a enters a capital "A".
^t	Abbreviation in tables for Shift+t or a capital "T".

1 In a word processor or any application with word wrap, there is essentially no difference between one line and one paragraph, so the terms are used freely in this paper.
2 It is also recommended that readers new to bidirectional text entry read: http://dotancohen.com/howto/rtl_right_to_left.html whose author approaches this subject differently.
3 See the second Design Note of this series "Exploring Complex Text Layout" for information about Contextual Forms and Shaping.

Basic Dual-Script Entry Example

Thai script, like Latin, is written from left-to-right. Entering "This is English and ภาษาไทย (literally: *the Thai Language*) is Thai" begins by typing Shift+t, which of course results in a capital "T." One big difference between Thai and English, though, is that Thai has no such thing as "capital" and "small" letters. That's actually fortunate; with forty-four consonants and a wide assortment of vowels (28) and tone marks (4), any keyboard that could support multiple cases for each character would be large and unwieldy. So, the shift key on a Thai keyboard layout selects between two different sets of characters: Pressing f for instance, results in the Thai ด consonant being displayed, while Shift+f produces the unrelated vowel โ.

Completing the example sentence that mixes Latin and Thai Scripts is quite straightforward, and consists of the making the following keystrokes:

Shift+t h i s spc i s spc Shift+e n g l i s h spc a n d spc ก 4 k Shift+k k w m p En spc i s spc Shift+t h a i .

Ignoring the first sixteen characters ("This is English "), the table below shows what happens – beginning with the "a" in the fourth word "and."

Character Entry and Disk Storage Order:	17	18	19	20		**21**	**22**	**23**	**24**	**25**	**26**	**27**		28	29	30	31	32	33	34	35	36
Keys typed on an English (US) keyboard:	a	n	d	space	ก	4	k	^k	k	w	m	p	En	space	i	s	space	^t	h	a	i	.
Character Display and/or Print Order:	17	18	19	20		21	22	23	24	25	26	27		28	29	30	31	32	33	34	35	36
Actual Characters Displayed and/or Printed:	a	n	d	space		ภ	า	ษ	า	ไ	ท	ย		space	i	s	space	T	h	a	i	.

Try entering this sentence by activating the Thai (Tis-820) keyboard emulator with whatever Input Method Editor you are using.

The top row shows the character count for our sentence, while the second row shows the keystrokes as entered. As long as both Scripts are oriented in the same direction – here left-to-right – nothing mysterious happens when mixing the different Scripts. The characters are displayed as expected (row 4), and stored in the computer's memory in the same order as they're displayed. The table above seems like overkill at this point. With a Thai keyboard layout chart handy (various options exist), typing k to enter a Thai า and Shift + k to enter ษ doesn't seem confusing. So what's the issue?

A More Interesting Dual-Script Entry Example

Hebrew Script, unlike Latin and Thai, is written, displayed and read from right-to-left. Intermingling both text directions in a single sentence is where things begin to get interesting – even confusing – and where "shortcomings" in many applications begin to appear. If we wish to write a sentence similar to that in the previous example, it would be: "This is English and שפה עברית is Hebrew." The English characters are entered and read from left to right, while the Hebrew characters included within the English block are read from right to left. ש is the *first* Hebrew character in the phrase.

For such a sentence, the required keystrokes look very similar to the Thai example (three characters "ew." at the end are eliminated to save space):

Shift+t h i s spc i s spc Shift+e n g l i s h spc a n d spc א a p v spc g c r h , En spc i s spc Shift+h e b r

This would seem to work exactly the same as the previous example and – up to a point – it does. But when switching text directions, things are just a little different; cursor movement in particular needs to be observed carefully to understand what's going on, so a more detailed look follows:

Typing “This is English and שפה עברית is Hebrew” – a step-by-step illustration:

Assume that we have already entered the first eighteen characters (“This is English an”) of the example bidirectional sentence containing Latin and Hebrew Scripts [4]. The cursor will be at position 19 as shown in step 1 on the right. Steps 2 and 3 show what happens when the “d” and “space” respectively are entered.

The initial English portion of the sentence is now complete. After each character was entered, it would have been equally valid to say “the cursor moved to the next character cell” or “the cursor moved to the right,” and both seem equally correct. It is important when discussing cursor movement, however, to understand that “next” is a relative term, and only in left-to-right Scripts does “next” mean the cursor moves “right.”

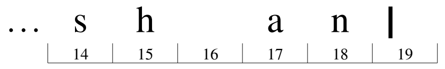
With the cursor ready to accept the twenty-first character, we “activate” the Hebrew keyboard (א) in step 4 in order to type the “ש.” Our sentence, however, is not yet bidirectional! Nothing has changed. Keep in mind that, unless and until at least one right-to-left character has been entered, we don’t yet have a bidirectional paragraph.

When we do enter the ש character in step 5 – using the a key – some applications will replace the existing cursor with a “bidirectional cursor” (usually a flag ⌝ or similar glyph indicating the current direction of text entry as shown in the examples). The cursor remains in the same cell, however, with the ש having been placed to its right.

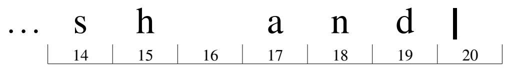
If the application offers such a bidirectional cursor, it will remain visible anywhere within that paragraph as long as a character with an opposite orientation is present. To avoid confusion, don’t do this now, but if we were to move such a cursor into the English portion, the cursor will appear as ⌜. We’ll get to cursor behavior in good time.

In step 6, when the two characters פ and ה are entered (using p and v), they shift to the right while the cursor appears to remain stationary – although it is now in cell 24.

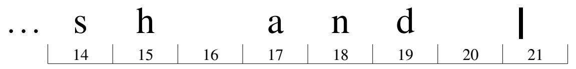
1. After typing the first eighteen characters, the cursor is in position to accept the next character – the “d.”



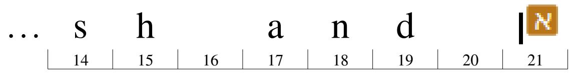
2. A “d” is typed, and the cursor moves to the “next” cell.



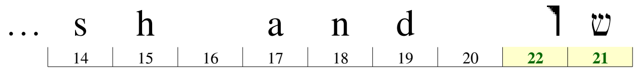
3. A space is typed, and the cursor moves once again.



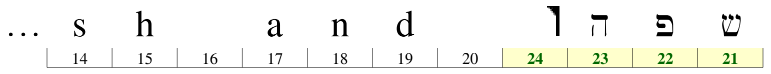
4. When א is selected. The cursor is waiting for the 21st character to be entered, but no changes are apparent.



5. When the ש is typed, the cursor now shows an RTL text direction but doesn’t move; ש is entered to its right.



6. As the פ and ה characters are typed, the cursor remains stationary as new Hebrew letters are shifted to the right.



The 24th character to be entered is a space. Along with tabs, line feeds and other characters collectively known as “white space characters,” the Unicode Bidirectional Algorithm[5] considers spaces as neither left-to-right nor right-to-left, but “neutral.” We’ll say more about “neutral” shortly.

4 In this case we are entering both English and Hebrew *Languages* as well, but the distinction between Script and Language is not that important for the moment. The “cells” are shown as being a fixed size here, but in practice – where fonts are usually proportionally spaced and character widths often vary – that isn’t always the case.

5 See http://www.unicode.org/reports/tr9/tr9-35.html for the official “Unicode® Standard Annex #9: UNICODE BIDIRECTIONAL ALGORITHM.”

At this point, we would reasonably expect pressing the space bar (spc) to result in the condition shown in step 7a on the right. We did not, after all, indicate any desire to revert to an English keyboard (using En) nor did we enter any left-to-right character. A typical typist would expect the cursor to be positioned to accept the next (25th) character (ע) of the sentence that follows the space.

But no! What often occurs looks similar to step 7b, a behavior that in our view must be considered – if not an outright Bug – at the very least a rather poor and distracting user interface design.[6]

User feedback during the entry of such “neutral” characters in many applications seems to have been designed by a group of sadists on a mission to frustrate all the writers, developers, translators, and others who simply want to type their text without having to learn any secret incantations. But, have patience – we will be divulging some of them on page 13.

Once we type the ע (with a g key), the display again regains its sanity and looks like example 8. Example 9 shows the display after entering the final characters ב, ר, י and ת [7] of the Hebrew phrase (using c r h and , keys). At this point, the En command (whatever that may be for your setup) will return the keyboard to its English layout, and the remainder of the sentence “is Hebrew” can be typed.

7a. When typing the space, a user might reasonably expect to see:

…	a	n	d		|		ה	פ	ש
	17	18	19	20	**25**	**24**	**23**	**22**	**21**

Note that 19 and 21 are adjacent characters, not 19 and 23!!

7b. … but instead, in many applications, sees:

…	a	n	d		ה	פ	ש		|
	17	18	19	20	**23**	**22**	**21**	24	25

Note the cursor is still showing RTL!

8. When the first character of the second word (ע - character 25) is entered, the cursor returns to the expected position, …

…	a	n	d		|	ע		ה	פ	ש
	17	18	19	20	**26**	**25**	**24**	**23**	**22**	**21**

9. … and entry of the remaining Hebrew characters presents no surprises. After the next space the cursor will mimic Figure 7.

…	d		|	ת	י	ר	ב	ע		ה	פ	ש
	19	20	30	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**

A common reason given for this odd behavior – when anyone can be found to speculate on it at all – is to maintain compliance with the Unicode Bidirectional Algorithm mentioned earlier. I don’t believe such an excuse stands up well to scrutiny however. In its introduction, Annex #9 states “*The Unicode Standard prescribes a memory representation order known as logical order.*” That logical order – the order in which characters are entered and stored in memory – is shown for our specific example in the top row of the following table:

Character Entry & Disk Storage (**Logical**) Order:	17	18	19	20		**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**		30	31	32	33	34	35	36	37
Keys typed on an English (US) keyboard:	a	n	d	space	ℵ	,	h	r	c	g	space	v	p	a	En	space	i	s	space	^h	e	b	r
Character Display and/or Print Order:	17	18	19	20		29	28	27	26	25	24	23	22	21		30	31	32	33	34	35	36	37
Actual Characters Displayed and/or Printed:	a	n	d	space		ת	י	ר	ב	ע	space	ה	פ	ש		space	i	s	space	H	e	b	r

*Enter these characters **in the order shown in the top row** using the “Hebrew (lyx)” or other suitable keyboard emulator available with whatever IME you use.*

6 In the Linux world, gedit, Kate, LibreOffice Writer, AbiWord, FocusWriter, and Calligra-Words all follow the perverse example in 7b. Gimp, Geany, FreeOffice TextMaker and Master PDF Editor behave in equally non-intuitive, although slightly different manners during bidirectional text entry. This evidence suggests that there are only a few common underlying services (HarfBuzz, Pango, CoreGraphics, DirectWrite, FreeType or combinations of these) being used by these applications to accept and lay out text.

7 To illustrate how even an experienced user can become frustrated, try typing ב, ר, י and ת – exactly as given here and above – including the commas; good luck with the order!

With some exceptions that are beyond the scope of this paper, *logical order* is typically the order in which a person who is familiar with the language would type the characters. When any of the applications mentioned in footnote 6 store the simple sentence above, the logical order on disk is correct. Annex #9 goes on to say: "*Each character has an implicit bidirectional type*" and that some characters are "*strong directional characters*" – "strongly left-to-right" (such as those in the Latin and Thai alphabets we used) or "strongly right-to-left" (e.g. those in the Hebrew phrase). It categorizes numeric characters as "*weak directional characters*" – a tacit recognition that most right-to-left Scripts order numeric values from left-to-right.[8]

Later, the paper says "*With the exception of the directional formatting characters,*[9] *the remaining bidirectional types and characters are called neutral. The algorithm uses the implicit bidirectional types of the characters in a text to arrive at a reasonable display ordering for text.*" A "neutral" space could therefore be considered as having either directionality. So why was the space (character 24) treated as a Left-to-Right character in example 7b and placed to the right of the initial word שפה? The common excuse seems to be that paragraph 3.3.5 of Annex #9 says "*Generally, NIs* [i.e. neutral and isolate formatting characters] *take on the direction of the surrounding text. In case of a conflict, they take on the embedding direction.*"

With only one "surrounding" right-to-left character, <u>there</u> <u>is</u> <u>no</u> <u>conflict!</u> The next paragraph is not considered[10] and, during entry, the "next" character may not even exist yet. The expected intuitive behavior of 7a <u>does</u> <u>not</u> <u>conflict</u> with Annex #9! When accepting new text, an application should consider "neutral" characters as having the directionality of the most recently typed "strong directional character" – which is the only "surrounding text" element. Further confirmation of this assertion can be seen in the example immediately preceding Annex 9's section 3.1.2, which makes it clear that the space between ה and ע in our example can and should be considered as a right-to-left character until the user explicitly declares it otherwise.

In preparation for the next example, confirm the behavior of the following four possible character entry sequences (designated α, β, γ, and δ for convenience) for our bidirectional sentence by typing the following sequences in separate paragraphs:

α. Shift+t h i s spc i s spc Shift+e n g l i s h spc a n d **spc** א a p v spc g c r h , En **spc** i s spc Shift+h e b r e w .

In this example (the sequence used above), keyboard changes are made as close to the right-to-left (Hebrew) segment/phrase as possible.

β. Shift+t h i s spc i s spc Shift+e n g l i s h spc a n d **spc** א a p v spc g c r h , **spc** En i s spc Shift+h e b r e w .

In the example above, both the left-to-right and right-to-left segments include the following space.

γ. Shift+t h i s spc i s spc Shift+e n g l i s h spc a n d א **spc** a p v spc g c r h , **spc** En i s spc Shift+h e b r e w .

In this example we effectively "surround" the right-to-left (Hebrew) segment/phrase with the necessary spaces.

δ. Shift+t h i s spc i s spc Shift+e n g l i s h spc a n d א **spc** a p v spc g c r h , En **spc** i s spc Shift+h e b r e w .

Finally, we could begin both the left-to-right and right-to-left segments with a space as shown above.

---

8 Once again, the reason for left-to-right layout of numeric values is out of scope here, but consider that science, math, and commerce aren't constrained by language or culture. Confusing the least significant digit with the most significant digit (e.g. reading 821 instead of 128) during activities or collaborations in any of those fields simply isn't tolerable. Numeric characters used in non-numeric contexts, such as within a part number, present other interesting challenges, but those are beyond the scope of this paper.

9 Directional Formatting Characters (the "secret incantations") are used to force certain directionality where it might otherwise be unclear; a few of those will be discussed later.

10 Section 3 (Basic Display Algorithm) and other sections of Annex #9 make it clear that the steps of the algorithm are applied separately to each paragraph in a document.

Editing an Existing Bidirectional Sentence – 1: Extending the Embedded Hebrew Text.

Hopefully, your text editor, word processor, or other application reacted the same with each of the four entry sequences (α, β, γ, and δ). Being consistent with entry of text segments that are laid out in opposing directions, while helpful in some ways, **may not help** to make any subsequent editing you might need to do any easier! To demonstrate why, we'll now attempt to edit each of the four sentences.

Change the Hebrew phrase in each of the sentences you typed from שפה עברית (meaning "Hebrew Language") to כתיבה ושפה עברית (which means "Hebrew writing and Language"[11]). Add the new phrase כתיבה ו to the beginning (right side) of the Hebrew portion (by typing f , h c v spc u) as shown in Exercise I on the right.[12]

Editing Exercise I:
Edit each of the sample bidirectional sentences (α, β, γ, and δ) to add the Hebrew phrase (כתיבה ו) as described in the text:

This is English and שפה עברית is Hebrew.

should become

This is English and כתיבה ושפה עברית is Hebrew.

You'll quickly discover that editing is – initially – not very intuitive. You might expect that a directional cursor would help, but unless you understand what's happening (actually what's *already* happened), editing existing bidirectional text can easily become frustrating. Although the next section illustrates a few ways to add the כתיבה ו phrase, it is important to attempt doing so prior to continuing; this is the best way to empathize with what many users will experience.

Editing an Existing Bidirectional Sentence – 2: Key Sequences for Extending the Embedded Hebrew Text.

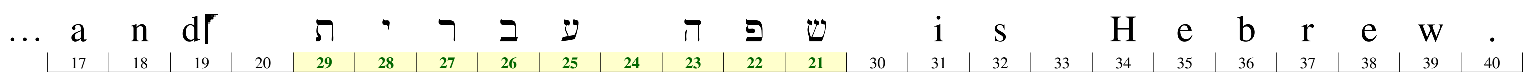
Place the cursor between the fourth word "and" and its following space, which is to say between the 19th and 20th character cell as shown below. It is necessary, of course, to switch to the Hebrew keyboard (א, if that hasn't already been done) before typing any of the additional characters.

The ⌈ cursor indicates that left-to-right text entry is expected (Step 10).

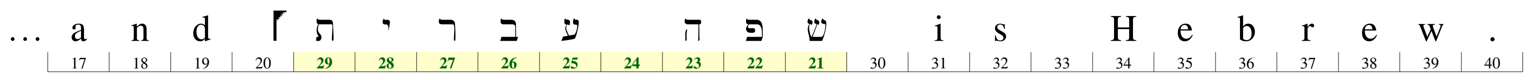
Press the right arrow key to advance the cursor past the space (Step 11).

Now type the כ character (using the f key). The כ has indeed been entered as the "next" letter, although it now appears ten positions to the right! Those who have never dealt with bidirectional text entry may not even notice the new כ.

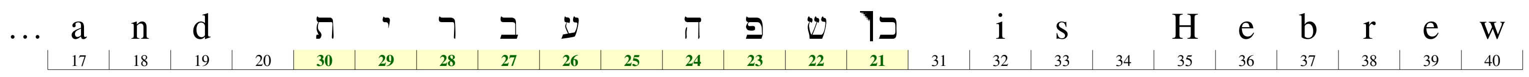
10. Place the cursor between the 19th ("d") and 20th (space) letters; then use the → key (*NEXT*, or the **Right** Arrow in LTR mode).



11. The cursor is now placed after the space, but still indicates Left-to-Right movement.



12. After the next (21st) character ("כ") is typed, the cursor now indicates a Right-to-Left direction; continue by entering " תיבה".



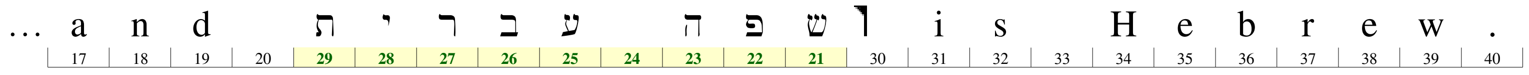
11 Note that we are using the capitalized version of the word "Language." The Hebrew translation of "language" is לָשׁוֹן, which is more or less equivalent to the English word "tongue" – not incorrect, but more generally meaning "speech" as opposed to a formal "Language" (our intent here). Language (capital "L") is expressed in Hebrew as שפה.

12 The lone ו character added with the u key to the beginning (right) of the word שפה changes the word's meaning from "Language" to "*and* Language" by the way.

The idea that the "next" character might not be the one immediately adjacent to the current one can be initially confusing. The key thing to remember is that the line of text being edited is the sequence of characters as they are stored (i.e. Unicode Standard's "*memory representation order known as logical order*"), not necessarily as they are displayed. Observe also that since the cursor in Step 12 is between the new "strongly right-to-left" כ character and the existing (also "strongly right-to-left") ש, the cursor has become a ⌉ rather than the ⌈ displayed in Step 11.

Note that the cursor in Step 11a is in exactly the same ***logical*** location as that shown in Step 11.

11a. This cursor indicates Right-to-Left movement, AND it is also placed between the space and the ש (the 20th and 21st characters).

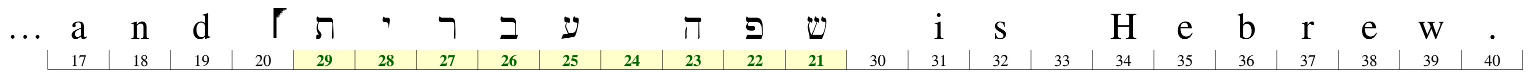


Because the cursor locations are equivalent[13], a second method of entering the new text segment beginning at the same location could therefore be:

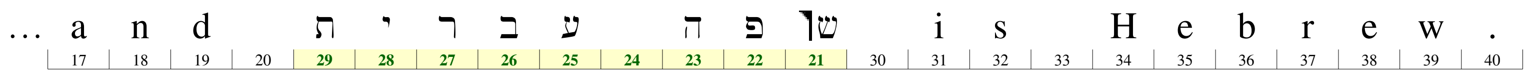
Using another copy of the original sample sentence, place the cursor as shown in Step 11 above.

Rather than typing the כ, as in Step 12, move the cursor forward as shown in Step 13. Again note that the cursor is now a ⌉ rather than a ⌈, since it is between the ש and פ – both right-to-left letters.

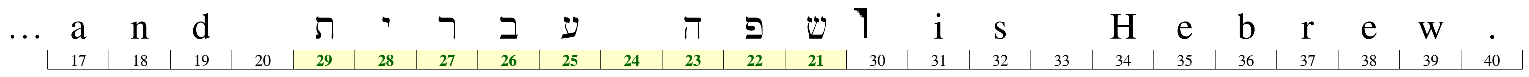
11. The cursor indicates Left-to-Right movement, and is placed between the space and the ש (the 20th and 21st characters).



13. Press the → key (*Next*, or the **Right** Arrow in LTR mode) and the cursor becomes positioned between the 21st ("ש" "the next character") and 22nd ("פ") characters – and changes to a ⌉ to indicate Right-to-Left movement.



14. Now use the ← key (***Previous***, or the **Left** Arrow in RTL mode) to move the cursor to the ***Right*** – before the "ש".



Step 14 now shows the same condition that existed in Step 11a and is therefore equivalent to Step 11, but set up for RTL input.

More Fun: In order to move the cursor to the *Right* in a right-to-left segment, we must use the *Left* arrow key, another behavior that can be counter-intuitive to users unfamiliar with commingling Bidirectional Scripts. Although this has been covered in an earlier Design Note[14], it bears repeating that – in a multi-Script world, the ← and → keys are better viewed as if they had the sense of the REV and FWD keys found on media players.

Finally, the new text segment could be added by placing the cursor within the existing Hebrew segment and moving it to the position shown in Step 14 with the ← key; in any case the new phrase כתיבה ו is added to the beginning (right side) of the Hebrew portion by typing f , h c v spc u.

We'll introduce two more examples of changes we might make to the original sentence (adding paired symbols, specifically parentheses and quotation marks), but before doing so, it will be helpful to demonstrate another behavior that can be confusing to new users: Word Wrap.

13 Ideally, there would be a simple "change text entry direction" command to move and change the cursor. We'll discuss this later in Secret Incantations on page 13.
14 See "Cursor Movement when Entering or Editing Text in Bidirectional Paragraphs" on page 14 of "Exploring Complex Text Layout" for more detailed information.

Word Wrapping Behavior with Bidirectional Text

Assume that we move the cursor to the beginning of the extended phrase we created in the last exercise in order to insert the phrase “Now is the time for all good men to come to the aid of their IT Department.” This is shown in Step 15; note that the bidirectional cursor (⸢) is indicating that left-to-right entry is expected. In this example, which you don’t actually need to type, the three Hebrew words are color-coded to add some clarity.

Step 16 shows the display after typing the first twenty characters of the new phrase. The “is Hebrew” portion of the sentence has wrapped to the next line as we would expect.

Step 17 illustrates that, once the next six characters (“all go”) are entered, the Hebrew portion of the text begins to wrap, but it is the word “עברית” immediately to the right of what we are typing that is wrapped, not the word “כתיבה” to the far right. By now, you should understand that this simply reflects the fact that “עברית” is the *final* word in the Hebrew phrase. For an uninitiated user, however, it often appears as if “עברית” is the *next* word (after “and”) – which of course it isn’t. Word Wrap is another perfectly logical behavior that can often be disconcerting until a typist adapts to the perspective needed when text direction shifts in the middle of an existing paragraph.

Steps 18 through 20 show how wrapping continues, with each Hebrew word sequentially dropping down to the next line.[15]

Word Wrap with Bidirectional Paragraphs

Begin with the extended sample sentence created above:

15. ⸢This is English and כתיבה ושפה עברית is Hebrew.

Observe the progression – particularly the word wrapping – as additional text is added to the beginning of the line:

16. … is the time for ⸢This is English and כתיבה ושפה עברית is Hebrew.
17. Now is the time for all go⸢This is English and כתיבה ושפה עברית is Hebrew.
18. Now is the time for all good m⸢This is English and כתיבה ושפה עברית is Hebrew.
19. Now is the time for all good men to co⸢This is English and כתיבה ושפה עברית is Hebrew.
20. Now is the time for all good men to come to the aid of their IT Dep⸢This is English and כתיבה ושפה עברית is Hebrew.

For the next examples, we’ll again use the original sample sentence “This is English and שפה עברית is Hebrew” to explore the behavior of paired symbols in multi-directional text. First we’ll add parentheses around the Hebrew segment; then we’ll edit the original text to enclose the Hebrew segment in quotation marks. The intent is to take two instances of the original sentence (α, β, γ, or δ, since we now know they should be identical) and change one to: “This is English and (שפה עברית) is Hebrew.” and the other to: “This is English and “שפה עברית” is Hebrew.”

The only change required for the first is to simply surround the Hebrew phrase with parentheses which, it turns out, are also “neutral characters.” For that reason, therefore, you might expect to encounter exactly the same issues you did when adding the “כתיבה ו” text above. Well, not exactly. As you shall see, you might find more success adding the parentheses, but for reasons that are not obvious. The second edit should be equally easy: add quotation marks (single or double) at the beginning and end of the Hebrew phrase.

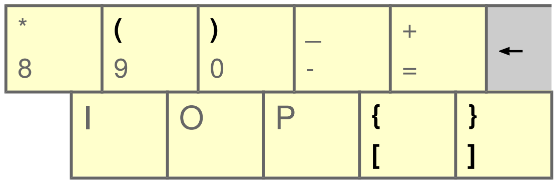
Before presenting either of these examples, however, we need to discuss something about paired symbols in general when editing bidirectional text. This momentary break is basically a summary/recap of “Paired Symbols in Text with Mixed Directions” in the second document in this series “Exploring Complex Text Layout” but is included here for convenience.

15 With Arabic Script, word wrap and justification can be even more interesting, but that isn’t in scope here. Remember that many Scripts may have distinctive behaviors.

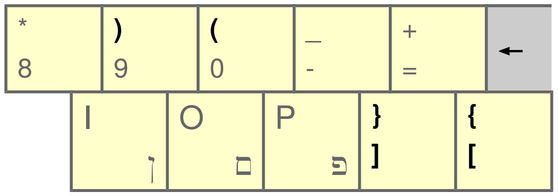
Adding Paired Symbols to Existing Bidirectional Text

Written languages may use a variety of paired delimiters: parentheses, braces, brackets, quotation marks, and so forth. With parentheses for example, the first such symbol in Left-to-Right scripts is typically referred to as either an "opening parenthesis" or a "left parenthesis." The second is similarly called a "right parenthesis" or a "closing parenthesis."

As shown in the sidebar, however, the "opening parenthesis" in Hebrew is a "right parenthesis.[16]" But, as the illustration shows, both Latin and Hebrew Scripts use the keystrokes Shift+9 to create their opening parenthesis and Shift+0 to type their closing parenthesis; the symbols produced in each case are, in fact, the identical character.[17] [18] As with the arrow key semantics and text layout discussed earlier, users need to be aware of the different perspectives in key naming when mixing scripts,[19] particularly when Input Method Editors are used to dynamically switch keyboard layouts.



Segment of an English (US) Keyboard Layout



Segment of a Hebrew (lyx) Keyboard Layout

The opening parenthesis in each Language uses the same keystroke (Shift+9), but results in a glyph indicating a closing parenthesis in the other.

Could anything possibly go wrong when these (characters) or their {[cousins]} on the row below are mingled in the same sentence?

For the paired symbol editing exercise shown to the right, the objective is to wrap the Hebrew phrase in parentheses. While it may appear that we need to concern ourselves with the philosophical question of whether the pair of parentheses is a pair of Hebrew characters *enclosing* the Hebrew text, or a pair of Latin parentheses *isolating* the Hebrew text, we don't. The Unicode characters are the same in either case. This makes things easier; we have twice the opportunity of succeeding in placing the character we want.

Editing Exercise II:

Edit one of the bidirectional sentences (α, β, γ, or δ) to add parentheses around the Hebrew phrase as described in the text:

This is English and שפה עברית is Hebrew.

should become

This is English and (שפה עברית) is Hebrew.

The next paired symbol editing exercise is to enclose the Hebrew portion of the sentence in quotation marks. If you completed Editing Exercise I, then II & III should present no difficulties.

Editing Exercise III:

Enclose the Hebrew phrase with quotation marks:

This is English and שפה עברית is Hebrew.

should become

This is English and "שפה עברית" is Hebrew.

It would be rather unfair to suggest attempting to type the sentence: "Here are some parenthetical phrases in English (the English Language), Yiddish (די ייִדיש שפּראַך) and Farsi (زبان فارسی) for your amusement." But if you're feeling confident, you may want to give that a try.

16 The Unicode Consortium originally used the names "opening parenthesis" for the "(" character `U+0028` and "closing parenthesis" for the ")" character `U+0029`, but later adopted the more neutral "left" and "right" shape-based terms as the official names, and demoting "opening" and "closing" to a lower status as acceptable alternate names.

17 … which is Unicode `U+0028`. Many languages also share these and other paired symbols, but place them in a variety of different keyboard locations.

18 As noted in "Exploring Complex Text Layout," Arabic (another right-to-left Script) keyboards produce left and right parentheses using Shift+9 and Shift+0 respectively, just as Latin keyboards do and the opposite of how Hebrew keyboards are laid out.

19 Many other paired symbols are saddled with names that make unwarranted assumptions about directionality. What does "right brace" or "left bracket" mean, after all?

Predominantly Right-to-Left Dual-Script Example

So far, the bidirectional examples have illustrated right-to-left segments embedded within a left-to-right sentence or paragraph. As mentioned earlier, since most modern languages display numeric values from left-to-right, a Hebrew sentence containing a count provides a simple example of the similarities and differences in data entry and editing when the primary directionality of a paragraph is right-to-left.

With the following sequence of key strokes in a new paragraph, we can type the sentence "ישנם 27 תווים באלפבית עברית" – which roughly means "There are 27 characters in the Hebrew alphabet." [20]

א h a b o spc 2 7 spc , u u h o , spc c t k p c h , spc g c r h ,

Character Entry & Disk Storage (**Logical**) Order:		27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	**6**	**7**	5	4	3	2	1
Keys typed on an English (US) keyboard:	א	,	h	r	c	g	space	,	h	c	p	k	t	c	space	o	h	u	u	,	space	2	7	space	o	b	a	h
Character Display and/or Print Order:		27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	6	7	5	4	3	2	1
Actual Characters Displayed and/or Printed:		ת	י	ר	ב	ע	space	ת	י	ב	פ	ל	א	ב	space	ם	י	ו	ו	ת	space	2	7	space	ם	נ	ש	י

Observe that it should not be necessary to do anything special to place the numeric characters "2" and "7" in left-to-right order; this should happen transparently if a right-to-left keyboard (in this case, Hebrew) is active and they are entered in the order given in the top row of the table.

Rule BD4 of the Unicode Bidirectional Algorithm's Annex #9[21] states that "*The paragraph embedding level*[22] … *determines the default bidirectional orientation of the text in that paragraph.*" Paragraph rule P2 in 3.3.1 says to "*find the first character of type L, AL*[23]*, or R*" and that "*the character found by this rule will generally be the first strong character after a paragraph separator or at the very beginning of the text.*" The sense of this section is that – with certain exceptions – if the first character in a paragraph has a right-to-left orientation (e.g. a Hebrew or Arabic character), the default alignment of the paragraph should be right-to-left. When the Hebrew sentence above is typed, what *should* appear is illustrated as follows:

Left Margin

If the following Hebrew sentence is entered at the beginning of a new paragraph or data field, it should of course be displayed from right-to-left with the exception of the left-to-right value "27." And it should be aligned automatically to the right margin as soon as the initial י character is entered: like this:

ישנם 27 תווים באלפבית עברית.

Right Margin

20 Of course there are, strictly speaking, only twenty-two Letters, but five of those have different forms when they appear at the end of a word. Using "22" (two identical digits) would certainly not be a great example of how numeric characters are reordered; "27" just makes a better example.

21 See footnote 5 for the location/URL of this part of the Unicode Standard.

22 This is the default directionality of a particular paragraph, and is also known as the "paragraph direction" or the "base direction."

23 For our purposes the "AL" can be interpreted as any punctuation specific to (and only used by) a right-to-left script, as distinguished from the more common punctuation characters "shared" with Latin. The "L" and "R" refer of course to "Strongly Left-to-Right" or "Strongly Right-to-Left" characters.

Secret Incantations

Although these are publicly available in Paragraph 2 ("*Directional Formatting Characters*") of the Unicode Standard's Annex 9 (see footnote 5), these are considered "secret" from the standpoint of a typical user, since we haven't encountered many data entry personnel[24] who have read that Annex.

You may have noticed that the Hebrew sentence displayed at the end of *Predominantly Right-to-Left Dual-Script Example* above contains a period at the end of the sentence, but the instructions for typing the sample do not. If you wish to attempt doing so, either by retyping the sentence entirely or by editing what you've already entered (and you now know that the techniques might be different), the Latin keyboard equivalent character when using a Hebrew IME keyboard is the `/` key. Whether typing `.` on a Latin keyboard or `/` on a Hebrew keyboard, the end result is a Unicode `U+002e`.

As it happens, you may well encounter the bug/behavior shown in Step 7b and find it difficult to place the period appropriately.[25] Fortunately, there are the Secret Incantations mentioned earlier. Although intended for special circumstances, we can use them to overcome certain bugs as well.

Secret Incantations (a.k.a. Non-Printing Characters) from Annex 9 of the Unicode Standard

Symbol	Code Point	Unicode Character Name	Description
LRM	`U+200E`	Left-to-Right Mark	A left-to-right zero-width character. Like Tab or Carriage Return "characters," it isn't displayed. (T א)
RLM	`U+200F`	Right-to-Left Mark	A right-to-left zero-width character. Like Tab or Carriage Return "characters," it isn't displayed. (Y ט)
LRE	`U+202A`	Left-to-Right Embedding	Treat the following text (until a PDF is encountered) as if it were embedded left-to-right.
RLE	`U+202B`	Right-to-Left Embedding	Treat the following text (until a PDF is encountered) as if it embedded right-to-left.
LRO	`U+202D`	Left-to-Right Override	Force following characters (until a PDF is encountered) to be treated as strong left-to-right characters.
RLO	`U+202E`	Right-to-Left Override	Force following characters (until a PDF is encountered) to be treated as strong right-to-left characters.
PDF	`U+202C`	Pop Directional Formatting	Ends the scope of the most recently applied LRE, RLE, RLO, or LRO.

LRM and RLM are called "Implicit Directional Marks" in the standard, and are the simplest of the "incantations" available. RLM is defined as a non-Arabic character; the ALM (`U+061C`) is the corresponding Arabic character. See Annex 9 for details.

LRE and RLE are called "Explicit Directional Embeddings" in the standard, and are used where the surrounding text isn't sufficient to permit a determination of the intended directionality. HTML entities for these are `&#8234;` and `&#8235;` respectively.

LRO and RLO are called "Explicit Directional Overrides" in the standard, and are intended for special cases; their use is otherwise discouraged by the standard as explained in Annex 9.

24 Actually, we haven't encountered many seasoned IT professionals who are familiar with this subject matter: hence the impetus for this introductory Design Note.

25 In various text editors tested (used primarily by IT folks), this works as expected. Unfortunately, it is quite difficult to add the period in many Office applications (used heavily by non-technical users) – an unexpected result. See the Note "Evaluating Applications for Bidirectional Text Handling" for an approach to evaluating your own portfolio.

PDF, which seems an almost perverse choice of acronym in this context, refers to the PUSH and POP operators originally introduced in assembly language programming – something else a typical user is likely unfamiliar with. Each of the preceding incantations remains in effect until the end of a paragraph or until this code point has been dutifully chanted. Its HTML entity is `&#8236;`.

Recall from the discussion of examples 7a and 7b that the cursor in some applications misinterpret "neutral" characters such as the period being entered here as left-to-right characters – actually they take on the overall directionality of the paragraph – if they are not followed by a character of the desired "strong directionality." When the same application also fails to automatically change the default directionality of the paragraph if the first strong directional character (in this example, the right-to-left י) differs from that of the page/document/entry field[26], the problem is compounded.

LibreOffice Writer and SoftMaker TextMaker, for instance, while both capable word processors, do not automatically change paragraph directionality unless the user explicitly sets it. Typing the following sequence in a new paragraph in either of these word processors:

Sequence 1: א h a b o spc 2 7 spc , u u h o , spc c t k p c h , spc g c r h , /

without first having explicitly set a right-to-left direction results in: ישנם 27 תווים באלפבית עברית.

Oops! The "neutral" period is interpreted as a left-to-right character. And, as we saw in example 8, it is only when a strongly directional character follows the period that it will revert to its proper position. Since we do not, however, have any intention of typing anything else in the paragraph, we can follow the period with the RLM incantation – adding a strongly right-to-left character that doesn't display. As it happens, the Hebrew (lyx) keyboard, like several other right-to-left layouts, has dedicated keys for both LRM (Shift+t) and RLM (Shift+y). So we can type:

Sequence 2: א h a b o spc 2 7 spc , u u h o , spc c t k p c h , spc g c r h , / Shift+y

without first having explicitly set a right-to-left direction, and the result will be: ישנם 27 תווים באלפבי עברית. with the period at the "end."

The Shift+t sequence can, of course, be replaced with other methods of entering a single Unicode value, such as those discussed in the "Character Entry Methods" section of the second Design Note in this series, "Exploring Complex Text Layout."

Applications that do follow the Unicode standard, and that automatically set paragraph direction based on the first strongly directional character typed into a paragraph, handle Sequence 1 without needing the RLM incantation. As mentioned, such applications include many text editors, but only one word processor – Calligra Words – we're aware of.

Although this situation will likely change as more systems become modernized to transparently handle the world's Scripts and Languages, there remain some details to overcome. LibreOffice Writer, for instance, properly stores the text above, including the RLM character (once the incantations are done or users train themselves to manually set the paragraph directionality) but we've found it quite difficult to retain the RLM when copying and pasting since there is no indication the character is present unless you pay very close attention to the bidirectional cursor.

This ends the second-last planned Database Design Note in the Multi-Script Database Series: "Exploring Bidirectional Text Entry and Editing."

26 See section "3.3.1 The Paragraph Level" of Annex 9 (see Footnote 5 on page 5)