Step by step instructions for

Reproducing Several Inter-Related Bugs

… all related to Script & Language-handling in LibreOffice Writer

Frank Oberle; July 2015

Preface. If segments of these instructions for reproducing bugs seem somewhat patronizing, please accept my apologies, but experience has shown that the entire subject area of Languages, Scripts, and Text Layout ('complex' or otherwise) is a mystery[1] to many, and over-explaining seemed the wisest course.

> **N.B.** For reasons that will become clear as this report progresses, my recommendation is to start from a known set of conditions; whether and exactly how the behaviors (bugs?) described below will appear depends on the state of far too many configuration options to do otherwise. For full disclosure, I should mention that my target for this report was a fresh Fedora installation in Oracle VirtualBox version 4.3.10_Ubuntu r93012, although I don't believe this to be relevant.

Initial (LO Default) CTL State: ☐ Complex text layout (CTL): OFF – Default CTL Language = Hindi

Step 1. Perform a new English Language installation of Fedora 21, which includes LibreOffice version 4.3.2.2.0+, Build ID:4.3.2.2-5.fc21. Although none of the behavior I'm describing is peculiar to either Fedora or any particular version of LibreOffice, certain aspects may be peculiar to Linux distributions.

Step 2. Note the icons/indicators at the top right side of the screen (illustrated here). Using Fedora's "Settings > Personal > Region & Language" setup utility (or your operating system's equivalent), use the [+] key to add the standard 'Hebrew' keyboard (there are typically multiple Hebrew keyboard layouts from which to choose in any given installation).

Figure 1 – Default Indicators in Fedora 21

> **N.B.** It is important to emphasize that this has added no support whatever for the Hebrew ***Language*** – it has merely added support for typing and displaying[2] the Hebrew ***Alphabet*** – an alphabet that is used for a number of different languages and happens to be written in a right-to-left (RTL[3]) direction. To be sure, these languages may use different elements of the alphabet, and may use them in different ways, but it is critical to understand that languages and character sets are quite different things!

Once the installation is completed, note that there is now another icon/indicator with the text 'en' and a down arrow. This indicates that an 'Input Method' (this example uses iBus) has been added, but that the original 'U.S. English' (en) keyboard remains active. So, for the moment, the keyboard will still produce the Latin characters printed on the tops of its keys.

Figure 2 – With New Keyboard Indicator

Obviously the details outlined above may be slightly different depending on the particular operating system in use and the particular Input Method installed, but I'll leave that for the reader to determine.

Step 3. Start LibreOffice Writer. Note that its default font is 12 point Liberation Serif.

Step 4. Using the 'Tools > Options... > Language Settings > Languages:' sequence in LibreOffice Writer, confirm that, under 'Default languages for documents,' neither the 'Asian' nor 'Complex text layout (CTL)' boxes are checked. Also note that the default selections shown for these, which are of course grayed out,

---

1 Since the word 'mystery' might seem unfair, a justification for this view is given in 'CTL Confusion' on page 20.

2 'Merely' is a bit unfair. There is far more involved in accomplishing this than some might suspect, but such arcana is irrelevant to reproducing the errant behaviors being described, so is out of scope – at least for the moment.

3 The implications of this and other fascinating and inconsistently defined and applied acronyms are discussed more thoroughly in the document 'Exploring_CTL.pdf' that should have accompanied this one.

☐ Complex text layout (CTL): OFF

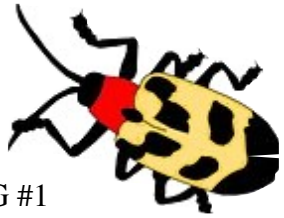
are 'Default – Chinese (simplified)' and 'Default – Hindi' respectively.[4]

Step 5. Return to the main document area and type a few words in English, then press the Enter key to begin a new paragraph. Using the keyboard, type "This is a mixed line. " (put at least one space after the period, but do not press the Enter key.) The cursor should be at the end of the new line.

Step 6. Using the keyboard selector button (Fedora's is shown in Figure 2 – you may also use whatever keyboard shortcut was assigned), change from 'en' (English (US)) to 'he' (Hebrew). The keyboard is now configured to produce Hebrew characters with no knowledge of their meaning, only an understanding that Hebrew script characters should be typed from right-to-left.

Step 7. Now, carefully observing each character as it is entered, press the a p , space g f r h and , keys in sequence. The following Hebrew text will appear once these keys have been pressed: שפת עברית. Pay particular attention to the entry sequence as each key is pressed. The ש character appears first, then פ appears to its left, and so on. Remember: no Hebrew *Language* support has been added to Writer!

We'll discuss the change in cursor shape later. When the space bar is pressed after the first , is typed, the cursor jumps to the right end of the Hebrew text, but then corrects itself when the following g key is pressed.[5] For now, ignore this behavior and continue pressing the key sequence exactly as instructed!



BUG #1
Premature exit from RTL layout

Step 8. After the final h key has been pressed, use the keyboard selector to change back from 'he' to 'en.' Once again, press the Enter key to begin a new paragraph. Continue to ignore the change in cursor shape.

While the Hebrew phrase you typed might just as easily have represented Yiddish[6], it was, in fact, '(the) Hebrew language' in the Hebrew language. To reiterate, neither the computer nor LibreOffice knows whether the *language* entered was Hebrew, Yiddish, or whatever. Without installing explicit language support, LibreOffice is unable (yet) to check the spelling (or grammar) of שפת עברית.

Step 9. Return the cursor to the paragraph containing the Hebrew text. Note several things here:

- No matter where you place the cursor in that "mixed script" paragraph, the cursor appears as what looks like a small flag rather than the normal cursor. The implication of that change will be discussed in Step 10; for the moment it's only important to notice the change.
- Insure that the flag cursor is in an English portion of the paragraph. The language indicator on the bar at the bottom of the screen shows 'English (USA)' or whatever flavor of English was chosen for the Fedora installation. You should also see that – unless you've changed any of the defaults –

4 These LibreOffice defaults seem arbitrary, given that '[none]' is available as a choice; it would be interesting to know why these particular languages (Chinese and Hindi) were chosen. But that isn't important.

5 This cursor 'burp' bug occurs because the default language in use here is English, which uses Latin script. Both Hebrew and Latin writing (among others) use the shared space character (u+0020). When the space is typed, Writer seems to incorrectly assume that the user has returned to using the Latin script. Once the change to the Hebrew (or any other) script has been detected, however, only the detection of alphabetic characters which are unique to the Latin script should trigger a return to that script. More cursor movement quirks will be discussed later, and also in 'Exploring_CTL.pdf'.
If a user is confused by Logical cursor movement (i.e. the actual character order), this can be changed to Visual (illogical?) with the 'Tools > Options > Language Settings > Complex Text Layout > Cursor Control > Movement: Logical | Visual.' Being a curmudgeonly sort, I find this akin to setting π=3.0 for those who are confused by decimals! But, 'de gustibus …'

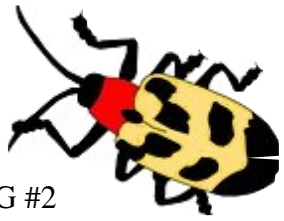
6 Although unrelated linguistically, Yiddish also uses Hebrew Script. Remember, changing the input method only results in the computer treating key strokes as if Hebrew script were being entered instead of the Latin script that was originally the default. The importance of understanding the difference between a script and a language is crucial.

the font in use is 12 point Liberation Serif.

- Using the 'Subset' drop down list of the 'Insert > Special Character...' dialog, note that the Liberation Serif font does ***not*** contain any portion of the Unicode Hebrew script/character block. Obviously, since the Hebrew text was displayed, the computer (at this point we don't care whether it's the operating system, the input method, or LibreOffice Writer) has found a source for the Hebrew glyphs it displayed for the Hebrew Unicode characters that were entered in step 7.

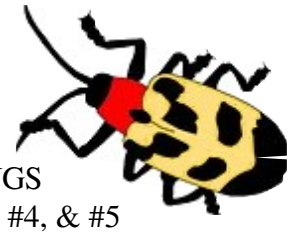
Step 10. Move the cursor into the Hebrew portion of the text; observe the following differences:

- Here, the "flag" cursor is now pointing in the opposite direction. In paragraphs with text that is displayed in different directions, this flag is a means of maintaining your balance as you get used to the cursor movement observed in Step 7, letting you know which direction the cursor will move as you press the arrow keys. Now, finally, the plot begins in earnest!
- With the cursor still in the portion of the text with Hebrew characters, notice that the language indicator on the bar at the bottom of the screen shows 'Hindi' – the first indication that something is amiss. Recall that we never selected the Hebrew *language* (or any other language for that matter), and we sure didn't select Hindi as either a script or a language! The displayed Language is incorrect! Whether or not this behavior is merely just one symptom of a deeper bug (which I believe circumstantial evidence suggests), it nonetheless qualifies as a Bug.



BUG #2
Incorrect Language Reported

- Now things get even more interesting; the font in use for the Hebrew text is shown as 12 point Lohit Devanagari[7]. Now it isn't surprising that a different font was substituted for Liberation Serif since, as we observed in Step 3 above, Liberation Serif does not contain any portion of the Unicode Hebrew block.[8] Using the "Subset" drop down list of the "Insert > Special Character..." dialog as we did in Step 9 however, you might be somewhat surprised to note that Lohit Devanagari doesn't contain any portion of the Unicode Hebrew block either. The font substitution indirectly reported to the user (well – if the user is astute enough to open the 'Insert > Special Character' dialog box) is incorrect[9], which certainly qualifies as a Bug!



BUGS
#3, #4, & #5
Arbitrary Font Substitution
Double Font Substitution
Incorrect Font-in-use Reported

- A little investigation (out of scope here[10]) shows that the font being displayed (or printed) is actually 12 point DejaVu Sans – which isn't indicated anywhere.

  We started with a serif font, and although the concept of serifs isn't actually relevant to most non-Latin scripts, the fact that the DejaVu Serif font wasn't used as the substitute is possibly due to the fact that the Serif version has no Hebrew characters (this is only mentioned in case it provides a clue as to what's going on). Many users, therefore, may not be able to determine what font is actually in use which, if not quite a Bug, certainly qualifies for many as an annoyance! For a discussion of why this group of bugs may not be caused – but merely exacerbated – by Writer, refer to footnote 9.

ANNOYANCE #1

7 Devanagari is another script used by many languages, among which is Hindi. This is technically known as a 'clue.'

8 i.e. the Unicode block u+0590 through u+05FF (hexadecimal equivalents of the decimal values 1,280 through 1,535).

9 See the section titled 'Is Linux Font Substitution Part of the Problem?' on page 31. This may also relate to Bug #38298.

10 If you're interested, save the file in *.fodt format and open it in a text editor to search for the fonts.

Conclusion #1. It seems clear that what appears to be the language setting[11] in 'Tools > Options... > Language Settings > Languages: > Default Languages for Documents > Complex text layout (CTL)' is used regardless of whether it is 'active' (checked) or not. Aside from the obvious confusion between scripts, alphabets, and languages[12] in this and other LibreOffice dialogs (and in much of the otherwise terrific documentation), it appears that this confusion likely permeates the code base as well.

☐ Complex text layout (CTL): OFF

Even without being activated it's obvious that only a single 'CTL language' can be defined at any one time. This limitation is Annoyance #2. It really should be Annoyances (plural) because CTL is actually a characteristic of Scripts – Languages only follow the scripts.

ANNOYANCE #2+

See LibreOffice TDF Bug 42123

Momentary Digression #1 for some background: The LibreOffice community seems well aware that many corporate and government agencies have begun adopting open source software. Indeed, such pronouncements are regularly posted in relevant forums, newsgroups, etc. What may not be as widely known is that some of these same entities have begun adopting – and even mandating – open source fonts for almost the same reasons in their 'official' documents.[13] The following paragraph, for instance, uses the preferred of thirteen Thai-English[14] fonts recommended for official government use in Thailand.

This is an 'official' Thai font (Sarabun) at 12 point. นี้ คือ ข้อความบางส่วน ไทย ตัวอย่าง. Notice that Sarabun's Latin is a Sans Serif.

As a point of interest, this document is using a 12 point FreeSerif body text; the difference in apparent character height between this font (we'll load it in Step 11) and the 12 point Sarabun font should be fairly obvious if you are at all sensitive to such things. Even more notably, all of Thailand's thirteen officially recommended fonts use sans-serif glyphs for the Latin alphabetic characters.

So, why no serif glyphs in the official Thai fonts? With Latin display and printing, we tend to evaluate our glyphs in terms like high or low 'x-height,' whether the glyphs have serifs or not, and a host of other criteria which are not necessarily comparable or even relevant to other writing systems. For example:

> The 'x-height' specification helps identify the differences between 'capital' and 'small' letters in the Latin alphabet but, while some Thai characters do appear taller than others (compare ก with โ for instance). Thai, like many other scripts, has no concept of 'capital' and 'small.' While identical in everything but their relative height, the Thai characters ร and โ may appear to be lower and upper case, but are completely different characters – the first being a consonant and the second a vowel.
>
> As for serifs, we can see that this A glyph has serifs while another A glyph for the same character does not. If we look at the first letter of the Thai alphabet (ก) and compare it to glyphs such as ถ and ภ, or even ฤ and ฦ, it might appear that serifs are present on these four Thai examples, but these are actually entirely different characters. (you can type these variants with the d, 5, 4, A and ? Keys once the Thai input method has been loaded, which we will do in Step 12.)

A
A

11 … and I say 'appears' only because a spell-check icon will appear if an appropriate language dictionary is installed.

12 Since you may ask: A script is a writing system, while an alphabet is a collection of letters using a particular script. A particular language may be written using one or more Alphabets. These distinctions are described in more detail under 'Languages, Scripts, Alphabets, & Locales' on page 33. These distinctions are not as clear as they could be in LibreOffice.

13 A particularly interesting example for the Thai language can be found at http://en.wikipedia.org/wiki/National_fonts and is worth reading to see the balance between achieving full internationalization while retaining a clear national identity and avoiding the often confusing and expensive licensing fees of commercial providers. Sound familiar? This is not unique to the Thai government, of course; it certainly isn't difficult to find parallel stories with other governments.

14 Virtually all 'official' fonts, whether sanctioned by government, academia or industry, include both English and any relevant national script(s). This is quite simply because English is required for operating systems, programming languages, etc. Many such officially sanctioned fonts intended only for local use contain only their jurisdiction's script and English.

With this limited introduction to some aspects of intermingling writing systems, we can begin to demonstrate even more insidious behaviors exhibited by LibreOffice Writer – keeping in mind that these may likely be symptoms of the same bugs reported earlier in this document. We'll accomplish this by describing some typical scenarios encountered when using multiple scripts (not necessarily even multiple languages – which present their own difficulties) in a single document.

Step 11. Download at least the basic Serif version of the GNU FreeFont and the standard version of the Thai THSarabun font[15] and install it as prescribed by your operating system. Many Linux systems, including Fedora, automatically install fonts when they are unzipped; otherwise you're on your own.

The purpose of using these specific fonts is simply to establish a common environment. The FreeSerif font (used in this document) has a wide variety of Unicode character blocks available that are reasonably well-designed and reasonably well-matched stylistically – including consistent character sizes for any given point size. The Sarabun font will be used later to demonstrate the actions required to match the heights of the 'official' Thai characters with fonts from other scripts when these need to be used together.

Step 12. Using the instructions from Step 2 above, add keyboard support for Thai (tis820 (m17n)). To test that this is working properly, begin a new paragraph and select the newly installed FreeSerif font at 12 point. Enter some English text as described in Step 5.

Step 13. As in Step 6, change the Input Method's script from "en" (English (US)) to "ท" (Thai (tis820 (m17n)). The keyboard is now configured to produce Thai script characters with – remember – no knowledge at all of their language or meaning.

Step 14. Press and release the [key and observe that the Thai letter บ appears. Follow this by using the b key to produce the Thai vowel ิ. This vowel is not placed in the next position, but appears above the previous บ character – บิ. This is a simple example of what much documentation suggests is Complex Text Layout. The Thai word บิ means 'business' by the way; like Latin script, Thai is written from left to right.

☐ Complex text layout (CTL): OFF

We're *almost* ready to demonstrate some further annoyances. We'll begin by adjusting the Thai Sarabun font; our intent is to utilize the FreeSerif Latin script font for the English text and the Sarabun Thai font for the Thai text. With Thai script input still selected, type the keys o u h space 8 n v - h and v to create the beginning of the sample text below, i.e. นี้ คือ ข้อ. These few characters are all that's needed to illustrate another aspect of Complex Text Layout.

This is 12 point FreeSerif Latin mixed with 12 point Sarabun Thai. นี้ คือ ข้อความบางส่วน ไทย ตัวอย่าง.

Not, as you can see, a great match. To use any of the 'official' Thai fonts with a Latin serif font, it is desirable to automatically adjust the glyph height of one or the other.[16] In this case, the best option is to make the Sarabun slightly larger. Experimentation suggests that a 12 point FreeSerif with a 15 point Sarabun will balance the heights while retaining correct proportions for the glyphs in each script.

This is 12 point FreeSerif Latin in line with 15 point Sarabun Thai. นี้ คือ ข้อความบางส่วน ไทย ตัวอย่าง

Step 15. Recall (and feel free to confirm as shown in Step 4) that – at this point – Complex Text Layout has not yet been 'turned on.' Now pay close attention to the first and sixth *displayed* character positions

15 Available from from: http://www.gnu.org/software/freefont/ and http://ftp.psu.ac.th/pub/thaifonts/sipa-fonts/ respectively.

16 Although not relevant to this discussion, it should be pointed out that, at any given size, the Sarabun font's cell is actually *taller* than that of the FreeSerif cell. It is only the size of the displayed characters themselves that is smaller. This permits any of Thailand's thirteen 'official' fonts – including the old style and swash faces like Chamornman and Srisakdi – to be freely mixed while maintaining consistent line spacing. Well thought out, if not obvious to the casual user …

(shown above in red). In each of these, the character ่ – a Thai tone mark – has been used.[17] In position 6 it is used alone, but in position 1 it is used in conjunction with the vowel ◌ี. Vowels 'outrank' tone marks, so the tone mark needs to be relocated above the vowel. This also is an example of what many consider to be 'Complex Text Layout' – in this case the variable positioning of diacritics above/below the previous character. Before tackling the glyph heights, however, let's pause once again.

Momentary Digression #2 to ponder the state of the world: The next steps 16 through 18 are meant to illustrate how personal computers really handle the vagaries of what is called 'Complex Text Layout' in 2015 – and demonstrate that they often do so with no help from LibreOffice.

Step 16. While the Thai keyboard entry is still active, open a command entry box for the operating system; this is usually accomplished in Linux with the Alt+F2 key combination. Using the same key sequence o u h space 8 n v - h v, confirm that not only are the Thai characters entered, but the positioning of the vowel and tone marking diacritics is correct. (See Step 14)

Step 17. Clear (but don't close) the command entry box, and then change the Input Method's script to Hebrew (see Step 6 if you've forgotten how to do this). The first thing you should notice is that the cursor in the input box has relocated itself to the right edge. As with the earlier Hebrew example (see Step 7) type the sample key sequence a p , space g f r h. Confirm that the letters are displayed right-to-left.

Step 18. If the behavior observed in Steps 16 and 17 – without LibreOffice's CTL support involved – comes as a surprise, it may be enlightening to experiment with just these two limited aspects of CTL: directionality of the script (right-to-left versus left-to-right) and flexible (smart?) placement of diacritics.[18] Here are some results of such very superficial testing on my demo installation. You are encouraged to perform your own, since this information is very relevant to some of the later discussions in this paper.

Limited Testing of Three CTL Functions in Contemporary Personal Computers

	Right-to-Left Text Test (using Hebrew)	Diacritic Placement Test (using Thai)	Diacritic Order Correction (Thai)	
Latin Keystrokes for Testing:	**ap, gfrh**	**ouh 8nv**	**ohu 8nv** [F]	***N.B.*** *The behavior of diacritic placement and order correction also depends on the 'Options > Language Settings > Complex Text Layout > Sequence Checking > Use sequence checking' setting as well as on the CTL Language setting.* *Again, the results of such interactions are not always intuitive but due to complexity, are not considered here.*
Application tested:				
Command Box (Alt-F2)[19]	supported [A]	supported	supported	
Shell/Terminal (Ctrl+Alt+T)	not supported [B]	supported	not supported	
vi(m) (character mode text editor)	not supported [B]	supported	not supported	
GEdit (GUI text editor)	supported [C]	supported	not supported	
Kate (GUI text editor)	supported [C]	supported	not supported	
AbiWord (Word Processor)	supported	supported	not supported	
LibreOffice Writer (Word Processor) – CTL OFF	supported [D]	partial/buggy [D, E]	not supported	
LibreOffice Writer (Word Processor) – CTL ON	supported	limited support [E]	supported [F]	
LO Writer – CTL ON and Thai Dictionary Installed	not applicable	font dependent	supported	
Gimp (Image Editor)	supported	supported	not supported	
to be continued as far as you wish …				

17 Note that these are the third and tenth ***keys*** that you pressed. The Latin 'h' was translated into the ่ Thai tone mark. The first *displayed* character cell represents a combination of three distinct *typed* characters of the Thai alphabet.

18 Once again, remember that these are only ***two*** aspects of what is typically meant by 'Complex Text Layout.'

19 The shortcuts listed here may differ depending on O/S versions, distributions, or even on the Desktop manager installed.

Notes:

A: Support for right-to-left text (even mixed with LTR text) and diacritic entry order correction in the Fedora command box was somewhat unexpected when compared to the lack of support for either in a shell or terminal.

B: Because these applications are primarily associated with some aspect of lower level computer control (e.g. shell commands, programming languages, etc.), lack of support for right-to-left presentation isn't that surprising. This is sheer speculation on my part though.

C: These GUI Editors are often used for the same purposes as vi(m), so a guess is that right-to-left support in these applications is either because the GUI editors are commonly used for many other purposes, or that right-to-left support is a serendipitous side effect from some graphics libraries – perhaps even from the desktop in use.

D: The fact that Writer's CTL capabilities are turned off while both right-to-left text and 'smart' diacritic placement function properly has to be regarded as significant. Continuing observation suggests that this isn't a matter of the CTL On-Off switch being 'broken,' but that it may simply be redundant in many environments. Hold that thought!

E: Admittedly, the examples so far don't fully support my 'partial/buggy' and 'limited support' characterizations, but that's coming. Note that among this group of applications however, only LibreOffice implies that CTL is a capability that can (or should) be turned on or off. In others, if we can draw any conclusion from just 'complex' diacritic placement support,[20] it appears that 'complex' (or any other) writing is 'just handled' transparently and without fuss by the Operating System – as it should be in any modern system with claims to universal support.

F. Although Thai conventions clearly indicate that vowels 'outrank' tone markings and should therefore be placed closest to the base character, note that the two initial diacritics (o and u) have been entered in the 'wrong' order. Once again, the command box support for display reordering of such pairs was surprising. Note that Writer handles the order correction properly only if CTL support is checked AND Thai is selected as the CTL Language.

Momentary Digression #3: Bugs and Annoyances in Practice

Before presenting more bugs and annoyances, we'll consider three distinct scenarios and work through two of them in order. Each relates, of course, to script and language handling use cases and, at least in my view, are not at all contrived or 'fringe' use cases. These are:

Scenario One: This very basic scenario is apparently the one envisioned by the developers of the CTL design approach used in Star*, and inherited – I suspect unchanged – by Open*, and Libre* implementations.

Assume that we wish to utilize two different scripts[21] within the same document. The example we'll illustrate first is based on the circumstances outlined in Step 14. As you will recall, we wish to use our preferred Serif font for the English portions of our text and, in deference to our Thai readers, an 'officially preferred' Thai font for the first of several 'foreign' (i.e. not the default locale or user interface) scripts we'll be introducing in this document. Scenario One will be covered in Steps 19 through 21.

Scenario Two: This example continues by adding text in yet another script from a language that has no 'official' font, and for which we can use the Unicode glyphs in our default paragraph font. Scenario Two will be covered in Steps 22 through 30.

Scenario Three: In this example, we'll again use our preferred English font but, unlike Scenario One, we'll be utilizing the preferred fonts for additional 'foreign' Languages we're quoting. If this seems like a contrived situation that will seldom occur, consider that the document you are now reading contains examples of several non-Latin scripts. If this whole Gordian area of LibreOffice is to be eventually untangled,[22] the LibreOffice documentation team might likely run into such a requirement as well. Scenario Three will be addressed in Momentary Digression #4 but, due to Annoyance #2, with workarounds.

Now for some detailed explanations …

---

20 Of course, we *can't* logically draw such a conclusion! Where there's smoke, however, …

21 It needs to be emphasized that this might not necessarily mean that we care about 'support' for two different *languages*.

22 Remember that the Gordian Knot was never actually untangled. It was severed! There might be the germ of a design approach buried in this metaphor. Although that's not my place to say, that is what I will be proposing.

Scenario One. In this scenario, we will attempt to remedy the relative height differences between the 12 point FreeSerif Latin characters and the 12 point Sarabun Thai characters discussed in Step 14. This seems like an ideal use case for LibreOffice Writer's option to enable CTL.

☑ Complex text layout (CTL): **ON** – CTL Language = Thai ▶

Step 19. Return to LibreOffice Writer. Since we've already established that CTL doesn't need to be turned on for many of its capabilities to be put in use, it might seem pointless,[23] but using the menu sequence outlined in Step 4, check the 'Complex Text Layout (CTL)' box and choose 'Thai' from the drop down selection box.

At this time, do *not* install any Thai language support, including dictionaries or the like. The reasons are: a) we're not concerned with languages at the moment – only scripts, and: b) doing so won't affect the behavior being illustrated. We may eventually want full language support, but it isn't needed now.

The settings should now look something like those in Figure 3 to the right. In order to proceed with the next step, Writer requires that you exit and then redisplay the 'Options > Language Settings' panel.

Since this is only a relatively minor inconvenience, I'll simply classify it as another annoyance.

Once the Language Settings options are displayed, additional CTL options will appear. The one we want appears under 'LibreOffice Writer > Basic Fonts (CTL).'

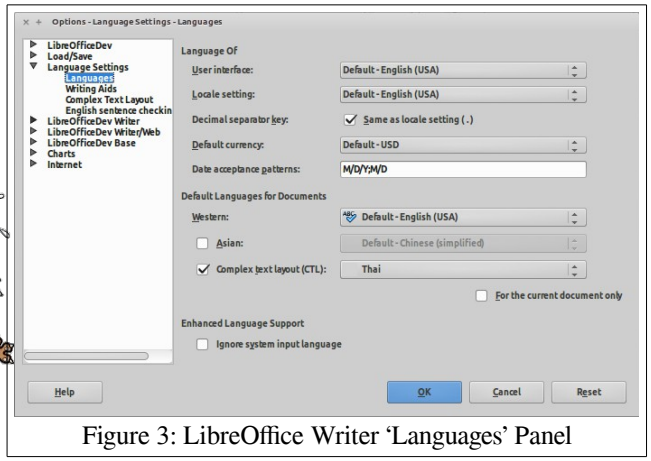
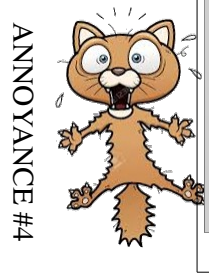


Figure 3: LibreOffice Writer 'Languages' Panel

Step 20. The panel shown in Figure 4 will appear. The first thing to do is to check the 'Current document only' box. Set each of the font choices (or at least the Default) to Sarabun, using a 15 point size for the Default to achieve the goals set in Step 14.



Having each of the settings on this panel offered independently gives a good deal of flexibility – certainly desirable in many circumstances, but my own experience suggests that having a 'master' setting at the top (which could be overridden by the individual choices) would be more useful in most uses. This might be something like 'Font' and 'Percentage' to be used relative to whatever appears in 'Basic Fonts (Western)' but this really just constitutes an enhancement request with likely a very low priority. (See Figure 11 for a sample).

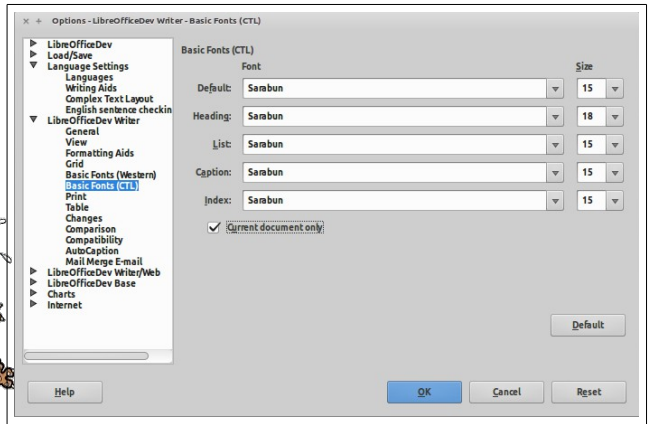


Figure 4: LibreOffice Writer 'Basic Fonts (CTL)' Panel

Step 21. Close the dialog and return to the document. Make sure that the English keyboard is selected and type 'This is some English text and ' (make sure to type the final space, but don't use the Enter key). Now switch to the Thai keyboard and press the key sequence o u h [ ] 8 n v [ ] - h v [ ] (once again, type the final space but don't use the Enter key). Switch back to the English keyboard and type 'is some Thai text in a similar size.' You should see:

23 Trust that this is all leading somewhere. Be patient!

This is some English text and นี้ คือ ข้อ is some Thai text in a similar size.

☑ Complex text layout (CTL): ON

If the size differences are not mitigated (compare this line to the sample in Step 14), exit and restart LibreOffice. When the CTL option is turned on, it is sometimes necessary to restart Writer for the CTL font and size selections to be recognized – this doesn't always happen, but I haven't been able to pin down the exact circumstances.

In accordance with our new CTL settings, Writer has automatically and conveniently altered the height of the Sarabun font's Thai script to match that of the default 12 point FreeSerif Latin script. Problem solved, correct? If you suspect that this question may be a trap, you're also correct.

ANNOYANCE #5

Is this LO Bug 66772?

Before proceeding to Scenario Two, mentioning another annoyance is unavoidable. If, while using an alternate keyboard that is *not for your CTL language choice* and type something you wish to undo using the Ctrl+z sequence (this, by the way, is not specific to LibreOffice; it's just more commonly encountered while typing documents), you'll discover that nothing happens. It may or may not take you long to discover that your highly trained instinctive hand motion is actually typing Ctrl+ผ (in Thai) or Ctrl+ श (in Hindi), not Ctrl+z.

Obviously, this caveat applies to a wide variety of shortcut key commands, particularly in applications such as LibreOffice. Quite a number of Writer users actually know how to type and, if you are a keyboard user, it can be really distracting and (oops, I forgot what I intended to say while correcting a Greek typo).

If any examination of this particular area of the code is undertaken, perhaps someone can come up with an imaginative approach to detecting such situations. Even a loud shriek from the speakers could help.

Scenario Two.

MEDICAL WARNING

Please skip Scenario Two and proceed to page 12 if you are prone to tirades, tears or temper tantrums.

We now have a document with the English portion of the text in our preferred serif font, and some Thai text in an 'official' height-matched Thai font. Next we'll add some Hindi text, which will be written in the Devanagari script. I'm not aware of any Hindi-speaking country that has specified an 'official' font[24] but, as it turns out, our default font has a fairly well-matched set of Devanagari characters, so typing हिनदी भाषा ('Hindi Language') should be simple, right? There is no need for a size adjustment, and we already know the steps to follow.

Step 22. As in Step 2, add one of the variety of Hindi keyboard options; for consistency, I recommend choosing the 'Hindi (Bolnagri)' layout, since that is what is used in these steps, but **don't** activate it yet.

Step 23. Do not open a new document! Making sure the English keyboard is still selected, type 'This is some English text and ' (make sure to type the space, but don't use the Enter key). Now switch to the Hindi keyboard and press the key sequence h i n d I [ ] B a S a [25] (as before, type the space but don't use the Enter key, and note that the second 'I' is capitalized). Switch back to the English keyboard and type 'is some Hindi text at the same point size.' You might reasonably expect to see:

24 The site http://fonthindi.blogspot.com/2013/11/official-hindi-fonts-used-in-ministries.html refers to a class of fonts as being 'official,' but all the Indian government specified was that any fonts used for official purposes must be Unicode fonts – i.e. the Devanagari characters must be located in the U+0900–097F character block. This site has a variety of information on Devanagari fonts, and also discusses some of the input methods used on Windows and Macs.

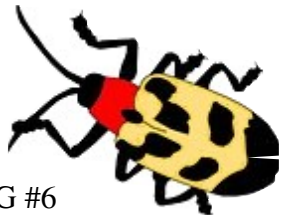
25 Yes, you just typed the word 'Hindi' but with what looks like an odd capitalization. Bolnagri is a phonetic keyboard layout, and what you just typed is actually the pronunciation (well, close enough) of हिनदी भाषा. **N.B.** You will notice that the second character typed is displayed before the first character – another CTL function based on some character ordering rules that are out-of-scope for the moment – but it is important to observe this to evaluate the results of the next steps.

This is some English text and हिन्दी भाषा is some Hindi text at the same point size.

Sadly, what you will more likely see is the much abbreviated version below:

This is some English text and ह भ is some Hindi text at the same point size.

Placing the cursor between the two Devanagari glyphs, you can confirm that the default 12 point FreeSerif font is used throughout – as we would expect, but only the initial characters of each Hindi word we entered (ह and भ) are displayed. We'll return to this quirk in Step 25, but first we'll review some other apparent results of having activated Complex Text Layout in Writer.



BUG #6
Typed Characters Lost/Ignored

☑ Complex text layout (CTL): ON

Step 24. Select the Thai keyboard once again. An even more fascinating (depressing?) discovery is that if we type our Thai phrase again (using the keys: o u h [8 n v), what is displayed is laid out properly: นี้ คือ, but with 12 point FreeSerif, not the 15 point Sarabun we would expect from the CTL settings. Even if you close and restart Writer, the Thai text will still be 12 point FreeSerif rather than 15 point Sarabun.

For clarity, repeat this Thai sample (นี้ คือ) in a new paragraph so you have two identical samples.

Step 25. Highlight the Thai text in one of the two samples you entered above. With the text highlighted, use the menu sequence Format > Clear Direct Formatting.[26] Observe that the Thai text there is now correctly set to the 15 point Sarabun we specified in the 'Basic Fonts (CTL)' panel shown in Figure 4.

WHAT ?!?: We didn't need to do that in Step 21 for the Thai CTL font (Sarabun) and size (15 point) adjustments to take place. So what's happening? Is this a Bug? An Annoyance? A misunderstanding? In the section 'CTL Confusion' on page 20, I mention that the otherwise helpful and thorough documentation for LibreOffice studiously avoids how all this (whatever 'this' turns out to be) works or is best utilized. At least some possible reasons or excuses for this otherwise mystifying oversight should now be apparent.

In any case, it *appears* that, when CTL was activated, LibreOffice Writer applied whatever font was specified to a new 'Custom Style' in Character Styles called 'Default Paragraph Font' – but this was applied to the text, not as a Style but as Direct Formatting. The new 'Default Paragraph Font' character style doesn't appear to 'inherit from' any other style, but seems to be created independently – perhaps copied from rather than inheriting from the Default Font specified in the 'Basic Fonts (CTL)' panel shown in Figure 4. Let's attempt to pin this down.

Step 26. Shift the cursor between the two sample Thai paragraphs, and note that the Paragraph Style is set to 'Default Style' in each example.

Right-click on the 'Default Style' entry in the 'Paragraph Styles' section under the 'Styles and Formatting' menu and choose 'Modify...' to display the 'Font' portion of dialog – shown to the right – to confirm that the default paragraph style was configured correctly. It certainly seems as if the original settings in Figure 4 were correctly applied to the Default Paragraph style.

Close that dialog and choose Character Styles from within the same menu.

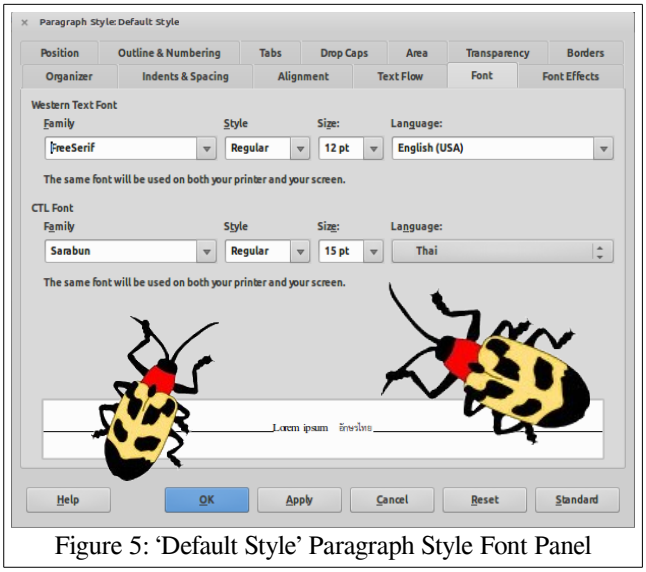


Figure 5: 'Default Style' Paragraph Style Font Panel

26 You didn't attempt to use the Ctrl+m shortcut to clear formatting, did you? I warned you. (see Annoyance #5 on page 9)

☑ Complex text layout (CTL): ON

Step 27. As can be seen on the right, the Character Style 'Default Paragraph Font' that was created seems to be properly set according to the earlier CTL Default Font settings. Close the dialog box.

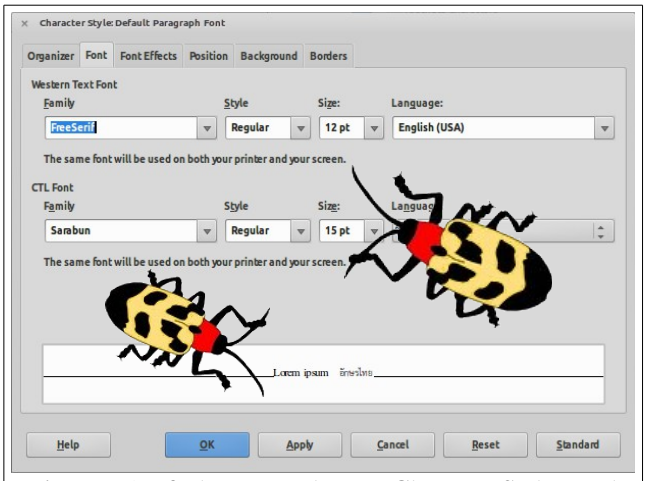
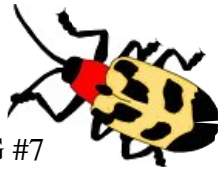


Figure 6: 'Default Paragraph Font' Character Style Panel

Move the cursor to the English/Latin portion of the text, and recheck the Character Style. Not surprisingly, it appears that this is set to the 'Default Style,' but the only option on its right-click menu is 'New...' The style exists, but apparently has no attributes set. (see sidebar in the table on page 6)

It also isn't mentioned anywhere that I could locate in the documentation why CTL Default Font Settings were applied to a *new* Style ('Default Paragraph Font') rather than to the existing 'Default Style' used by the normal (or at least corresponding 'Default Style') paragraph style. Could this be a source of the mysterious behavior? There seems to be another Bug in all this.


BUG #7
Wrong Style Assignment?

Step 28. Close any open dialog boxes and return the cursor to the Hindi phrase you attempted to type earlier in Step 23; it appeared – rudely truncated – as:

This is some English text and ह म is some Hindi text at the same point size as the English text.

Highlight the Hindi text and once again and use the Clear Direct Formatting command as you did back in Step 24. Nothing changes. Those seven characters have apparently joined Amelia Earhart and Jimmy Hoffa in the ranks of the permanently missing.

☐ Complex text layout (CTL): OFF ▶

Step 29. Return to the 'Tools > Options... > Language Settings > Languages:' sequence in LibreOffice Writer, and uncheck (turn off) 'Complex text layout (CTL).' Observe that the default selection shown, although of course grayed out, remains 'Thai.'

If we now begin a new paragraph, switch to the Thai input (as in Step 13), and type the test characters o u h 8 n v , we find that the result is นี้ คือ – with properly positioned diacritics, rendered in the default FreeSerif font – not the desired Sarabun. Neither the Font nor the Size settings established for CTL are in evidence. Now, for even more entertainment …

Step 30. Select the region with the Thai characters and again use the menu option 'Format > Clear Direct Formatting...' In a move that would make Penn and Teller proud, the นี้ คือ is magically restored to นี้ คือ with correct settings for both font and size.

A Possible Inference from the observed Behavior

LibreOffice ***seems*** to apply CTL settings when it encounters any characters from a non-Latin Unicode segment.

Regardless, with CTL turned on, the first Devanagari character (or character from a script not specified in the user's CTL settings) ***after*** a Latin character was displayed, but the remainder didn't appear. Could this be because the Sarabun font has no Devanagari characters? (which begs the question of why some glyph substitution wasn't applied …)

Thus, when 'hindI BaSa ' is typed, only the characters produced by the h and B – each of which are preceded by a Latin space (like many scripts, Devanagari shares some lower ASCII characters such as the space and tab) – are displayed. What's more interesting is that they are displayed at 15 point rather than at 12 point (following the user's CTL **size** settings). But they don't follow the user's CTL **font** settings, which seems rudely inconsistent.

The two initial characters are taken from the Latin default font (FreeSerif) that happens to have Devanagari characters) rather than disappearing as the others do.

Might this suggest that the CTL isn't triggered until *after* the first non-Latin character is encountered and, further, that the cursor position might be out of sync with the recorded input stream? And what happens if the default font doesn't have any characters from the Devanagari block?

Just wondering, of course …

Momentary Digression #4: Magic Workarounds for the Bugs and Annoyances

A number of workarounds exist and are widely recommended on various forums, most involving the use of styles. A few quite complex solutions involve macros that, while they might be quite entertaining as programming exercises, will likely never be adopted by the average novelist or accountant.

It is certainly possible to write a document that freely intermixes scripts and even languages; the evidence is in your hands. But even with what's been outlined so far, adapting to these quirks is far more tedious than it should be.

The most often recommended workaround for many of these quirks is to create separate character styles for each script (language) in use; some recommend leaving CTL on and some don't.[27] This can be made to work, but seems similar to suggesting that a user create separate character styles for Bold (using the FreeSerifBold font), Italic (FreeSerifItalic) or Bold-Italic (FreeSerifBoldItalic).

And then, of course, the user might need to set separate styles for each variant of the default font – Headings, Contents, Numbering, and so forth. Sorry; NO! That's simply not acceptable in a contemporary piece of software. In fact, most software doesn't even list the bold and italic variants; it just applies them as required. And such variants aren't usually even listed on the font menu.

Momentary Digression #5: What does 'For the current document only' really mean anyway?

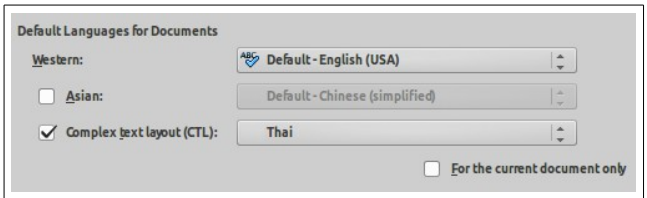


Figure 7: Detail of LibreOffice Writer Languages Panel

To the right is a closer view of the 'Default Languages for Documents' section of the Tools > Options > Language Settings > Languages dialog.

Primarily due to its horizontal placement,[28] the check box labeled 'For the current document only' can easily be construed to suggest that the 'Asian' and 'Complex text layout (CTL)' support as well as specific language selections on this panel are set for the current document only. In fact they are not. To be fair, the description in the LibreOffice Version 4.2 Writer Guide (the current version) makes the actual operation almost (but not perfectly) clear.

> "If you want the *language* setting to apply to the current document only, instead of being the default for all new documents, select For the current document only." (Chapter 2, page 67; the italics are my addition …)

Contributing to this potential misunderstanding is the use of the singular word 'setting' rather than the plural 'settings,' since it (so far as I can tell) applies to everything on the panel[29] except for the 'Asian' and 'Complex text layout (CTL)' check boxes. But it might not, as suggested by the following:

> "On the right-hand side of the Language Settings – Languages page (Figure 61), change the User interface, Locale setting, Default currency, and the settings under Default languages for documents as required. In the example, English (USA) has been chosen for all the appropriate settings."

… which implies that the 'Date acceptance patterns' remain universal. Although it isn't that difficult to figure this out, when coupled with the behaviors outlined above, attempts to demonstrate Writer to new

---

27 Lack of universal consistency in such matters is usually always a characteristic of complicated (as opposed to merely complex but not yet understood) underlying code. As I hope has been demonstrated by now, different results will occur depending on whether CTL is on or off, and not in the way many users might expect.

28 Specifically its close proximity to the row above as well as its horizontal offset. At least if its left were aligned with the left sides of the other language blocks, the scope of its actual purpose would be *slightly* more suggestive.

29 … which include User interface, Locale setting, and Default currency.

users (as in FOSS conversion attempts) don't always go well.

So I believe the design and layout shown in Figure 7 together constitute another annoyance. This may actually be mitigated without a major overhaul but, again, might not really have much impact on the issues experienced by those who utilize multiple scripts, much less multiple languages. While the dialog shown in Figure 7 is open, let's look at one more Bug.

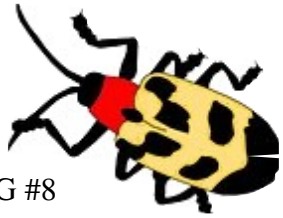
Another serious flaw exposed by the Figure 7 dialog has been the subject of much past discussion. A typical and rather succinct statement of this was made by the poster of TDF Bug 42123 in October 2011: 'it is wrong to group scripts as "Western", "CTL" and "Asian" in the font selection dialog.' Feeling perhaps that this might simply have been an opinion, or might simply be a reflection of cultural sensitivity,[30] developers relegated this bug (along with other similar bugs) to the 'CLOSED – WONT FIX' category. Fair enough, but I believe this is in fact a logical flaw with some actual usability consequences, not simply a matter of opinion or oversensitivity.

LibreOffice Writer's two choices, 'Asian' and 'Complex Text Layout (CTL),' do not constitute internally consistent and rational domains from which a single choice can *reasonably* be chosen. While one might argue that this is mitigated by the ability to select both, various intersections between the two domains prevent any reasonable conclusion from being drawn by a user as to what effects might be expected.

> **Pick One or Both:**
>
> Would you like to have LibreOffice developers who are:
>
> ☐ Tall, blonde, under age, and fluent in C++, Smalltalk, and German, or
>
> developers who are:
>
> ☐ Smart, short, retired, and fluent in Ada, Arabic, and Java?

Like the choices 'Asian' and 'Complex Text Layout (CTL)' in Figure 7, neither of the choices on the right are logically defensible groups of characteristics. What happens if you need both C++ and Java skills?

Would you end up with developers who are both tall and short? Is Hindi not used in Asia? Is handling of Chinese not as complex as handling Thai? For brevity's sake, however, further discussion of Bug #8 will be curtailed. But:



As I believe has been demonstrated so far, the effects of selecting from two discrete groupings of arbitrarily assembled characteristics can border on mystifying to many users.[31] Such collections of behaviors in each of these arbitrary categories comes as an 'all or nothing' proposition, which isn't always desirable for particular uses. Now let's look at one final[32] Bug before closing:

One Final 'Justification' for Reworking Writer's Script and CTL Handling …

On page 82 of the LibreOffice Version 4.2 Writer Guide (the current version), the terms in the illustration refer correctly to 'Align Left,' 'Centered,' 'Align Right,' and 'Justified.' These line layouts are also referred to colloquially as 'left justification' ('flush left' or 'ragged right'), 'right justification' ('flush right' or 'ragged left'), and so forth. Here, however, the bug we are illustrating deals only with 'real' or 'full' Justification.

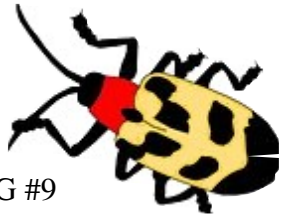
In its precise printing sense, justification refers to laying out the text so that it fully occupies the linear space between the left and right margins. This is typically accomplished by adding spaces between

30 In the note at the end of Norbert Thiebaud's 19 June 2012 posting, he comments '... I'd prefer if the discussion center on practical and technical aspects rather than politically correct ones.' I'm not at all unsympathetic to that view generally but, this example has far less to do with political correctness than logic.

31 And, as discussed in CTL Confusion on page 20, I'm not at all sure this isn't true for many developers as well.

32 It's not the final bug related to language and script handling – I'm getting old and it's just the last bug I feel like discussing.

words,[33] between character cells,[34] or a combination of both. Generally, the final line in fully justified paragraphs – which would often look quite bizarre if stretched across the entire width[35] – is set as if it were flush right or left depending on the normal text direction of the script.



BUG #9
Full RTL Justification Broken

LibreOffice Writer, however, currently sets the last line improperly when it performs full justification on the right-to-left (RTL) scripts I have used. We can demonstrate this with Steps 31 through 34.

☐ Complex text layout (CTL): OFF

Step 31. Writer's CTL support should still be disabled from Step 26 above. Verify this and begin a new paragraph. Don't open a new document (yet), since we wish to retain the existing default font settings, etc.

While previous examples offered in this document to reproduce bugs didn't need to rely on more than a few characters, full justification requires more text than you will likely wish to type by hand. So, in the spirit of universality espoused by the Document Foundation's 'TDF Manifesto,' we'll turn to another body espousing the same spirit to gather some sample text.

The Universal Declaration of Human Rights didn't have anything to say about being nice to the world's beleaguered software developers, but Article 1 of that manifesto has been translated into over 300 languages,[36] thus providing coders with a good source of text that can be used when demonstrating or evaluating how fonts, scripts, character sets, languages, etc. are handled in any environment.

Open the site referenced in footnote 36, and proceed to Language Families > Afroasiatic > Hebrew. Locate the standard Hebrew version[37] of Article 1, and copy it into your Writer document. By now, you should be familiar enough with cursor movement in right-to-left scripts to avoid getting dizzy.

Caution: In the event that you're feeling adventurous and wish to load an Arabic version of Article 1 as your example right-to-left script instead of Hebrew, please restrain yourself. Although Arabic script exhibits exactly the same problem with handling the final line when fully justified, there are some further refinements used in Arabic justification[38] that will simply be a distraction here.

Step 32. Because the Article 1 snippet is not very long, it will be helpful when looking at full justification to adjust the paragraph margins so that the text is spread over at least three lines. In the example below, both the left and right margins have been given an additional indent of one inch each. Experiment!

Now paste the copied text into the document. You will see something like the following:

ל בני האדם נולדו בני חורין ושווים בערכם ובזכויותיהם. כולם
חוננו בתבונה ובמצפון, לפיכך חובה עליהם לנהוג איש ברעהו ברו
ח של אחווה

33 Of course, this can be difficult for languages that don't use spaces between words, such as our second CTL example: Thai.

34 Not simply characters; the distinction is discussed in Character Codes and Character Cells on page 39.

35 This description is of course related only to horizontal (left-to-right and right-to-left) scripts. My impression is that justification isn't a concern for vertical scripts because their character/glyph heights are matched, but I have no first-hand experience with such texts, so it might be worth researching if any redesign is undertaken.

36 See www.omniglot.com/udhr/index.htm – this contains links to the text of Article 1 of the United Nations' Universal Declaration of Human Rights translated into over 300 languages. It is also a good source for observing layout behaviors.

37 The browser I use is Firefox, but any contemporary mainstream browser should display these samples correctly.

38 Those refinements are known as Kashideh, which is described and demonstrated in the Exploring_CTL.pdf document that accompanies this one. Aside from the 'last line' bug, Writer appears to be capable of handling Kashideh well but, as Arabic is certainly not a specialty of mine, this should be independently verified.

Notice that the final line of text is set flush left rather than flush right, which is typical of RTL scripts.

Apparently, the differences in justification between RTL and other layouts might not ever have been considered by the original developers – the text from page 83 of the Writer Guide simply says:

> "When using justified text, the last line is by default aligned to the left."

Although somewhat of an annoyance, it might seem that we can at least correct this manually. Concerning the alignment of the final line, the Writer Guide goes on to say:

> "These options are controlled in the Alignment page of the Paragraph dialog, reached by choosing Format > Paragraph from the Menu bar, or by right-clicking in the paragraph and selecting Paragraph from the context menu, or by clicking the More Options button on the Paragraph panel in the Properties deck of the Sidebar."

Nope. No such luck. As can be seen in Figure 8 on the right, the choices presented there for handling the last line are limited to 'Left,' 'Centered,' and 'Justified.' But no 'Right.'

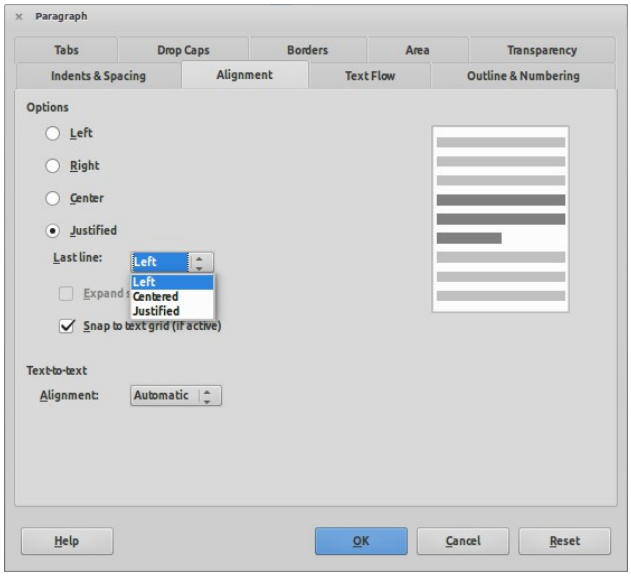


Figure 8: Writer's 'Paragraph Format' Alignment Panel

Once again, there is a commonly encountered work-around. RTL text requiring full justification can be placed into either a table cell or frame, set to full justification (of course), and with the internal alignment of the cell or frame set flush right.

Is such a workaround really acceptable in a modern word processor claiming to support multiple scripts and languages? Is setting otherwise straightforward text into frames or tables considered convenient?

But – perhaps this is one of the adjustments that depend on setting the CTL 'language.' It would be remiss not to test this by changing the CTL settings.

Step 33. Turn CTL ON again (using the menu sequence outlined in Step 4), but this time after checking the 'Complex Text Layout (CTL)' box, choose 'Hebrew' as the 'Language' from the drop down selection box. Using the close-reopen process described under Annoyance #3, and the dialog displayed in Figure 4, change all the fonts from Sarabun – which only contains Latin and Thai glyphs – to FreeSerif. which we know has reasonable Hebrew characters. The font size selections are of no interest at this point.

Step 34. Begin a new paragraph and repeat the process of pasting the Hebrew text sample into the document and setting the paragraph to be fully justified. It will quickly become apparent that nothing at all has changed regarding Writer's handling of full justification. Be sure to open the dialog illustrated in Figure 8 and confirm that 'Right' still isn't available as a choice for the 'Last line.'

As an incidental observation, recall that LibreOffice's justification algorithm(s) – aside from this right-to-left bug – is thought by others to be in need of some attention anyway, so this particular quirk might be addressed if that algorithm is given any attention.[39]

☑ Complex text layout (CTL): **ON** – ▶ – CTL Language = Hebrew

39 Document Foundation Bug 38159 (Better full text justification with auto character scaling and paragraph level adjustment) has been listed as 'NEW' since 10 June 2011, but comments continue to be added as of 20 May 2015.

Momentary Digression #6 to ponder some cautionary history: For those who haven't noticed, we're rapidly approaching the latter part of a prolonged shift in how languages, alphabets, and fonts are handled in modern computers, and it might be helpful to remember a similar shift from a quarter century ago.

The history of printer support in applications, particularly in word processors, should serve as a cautionary tale to the current state of language and font support in LibreOffice, and particularly in Writer. While this digression into printer support history may initially seem somewhat off-topic, its relevance should become clear as we continue.

> In the latter half of the 1980s, it was mostly a DOS world for personal computers. My own organization had a rather interesting product from Microsoft called "Windows" installed on two of the PCs in our electronics test facility, but only because it came with and was required to support graphic interfaces in LabView – an application used to control our electronic test equipment set-ups.[40] WordPerfect and Lotus 1-2-3 were the products of choice for 'office' applications. We had dot matrix printers and a rather expensive (and ridiculously heavy) LaserJet for documents which served to impress others with our professionalism. Most applications of that era had their own printer drivers (the days of manually inserting 'Bold ON' and 'Bold OFF' codes into documents – and perhaps altering them to print on a different device – had passed); each printer either had one 'font' (often 9 dots by 6 dots) or a set of built-in fonts that came in a few specific sizes.
>
> In the early 1990s, Windows 3.0 appeared to great hype, with pretensions of being an actual operating system; shortly thereafter, printer drivers began to appear in this 'new' OS. What a terrific and obviously rational decision that seemed to be! Except ….
>
> Except the drivers offered by Microsoft were fairly pathetic; an updated WordPerfect for Windows dutifully began offering the user a choice of using either the Windows drivers or its own far more complete (both in terms of accessible printer features as well as number of models) drivers. WordPerfect, after all, had offered its word processing software on pretty much any available platform (including most mini-computer operating systems) for many years.
>
> But having the operating system provide the drivers was the correct and sensible approach – it meant any application developed for the O/S had a consistent printing API and so on. Programmers didn't need separate routines for different classes of printers, and the printer manufacturers had a standard target to aim for. Migration of printer support to operating systems was inevitable.
>
> As printers matured, and fonts themselves began to be both scalable and independent of the printers, the transition to the world we are now used to occurred rather quickly. Within a very short time, individual applications ceded management of printer drivers to the operating system.

So how does this nostalgia relate to how languages, alphabets, and fonts are currently handled by applications (and specifically by LibreOffice)?

Consider that, even ten years ago, the idea of having one standard code for each character in each alphabet of each of the world's living, contrived, and abandoned writing systems seemed pretty impractical. Writing in more than one language in a single document often required jumping through hoops – and it was often much easier to simply acquire a separate word processor for the variety of languages that had

<< BREAKING NEWS >>

CODE PAGES SURRENDER!

The use of Unicode has won the character-definition war, while the skirmishes among the various bit layouts for representing these codes have also ended– with UTF-8 widely acknowledged as the clear winner.

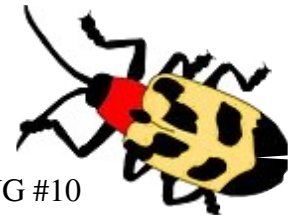
Get with the program …

40 LabView, originally designed for the Apple Mac, is still available today. According to WikiPedia, a DOS/Windows version wasn't released until 1990, but we were certainly using it quite successfully in 1988.

'interesting' output requirements.[41] Like the idea of moving printer drivers into the operating system, adopting Unicode seemed to remove much complexity, and its adoption accelerated once UTF-8 came along. Writer needs to catch up with the industry's migration of 'interesting' text input handling (as well as the so called 'Complex Text Layout' varieties) from applications. For the most part, the handling described for languages such as Thai and Hindi is already present in operating systems that have been introduced in the past five (and often more) years. Having CTL as an on-off option in LibreOffice isn't even necessary, and should only be loaded and made available on older operating systems.

So what's the message here? It isn't an oversimplification to say that the support architecture for scripts and languages in LibreOffice has some genetic flaws inherited from its grandparents. The definitions of `CharacterProperties`, `CharacterPropertiesAsian`, and `CharacterPropertiesComplex` have been part of the API at least as far back as the StarOffice days. StarWriter's original non-western support was based on 12 specific languages with no pretense of being a generalized solution. Although this grew to not quite 26 languages[42] by the release of StarOffice 7, one suspects that these may have been added in what is charitably called an 'ad hoc' fashion with no overall architecture or design. Certainly the lack of distinction between scripts, alphabets, and languages remains evident in both the LibreOffice dialogs and documentation today. Sun purchased StarDivision in 1999, which means the architecture has been unchanged for at least sixteen years, and likely longer. In the world of computers and software, that's a long time.

Step 35. One last, final (really, this time I promise) bug illustrating the general innate bias for left-to-right script in Writer is that the recognition and proper handling of right-to-left script layouts is only applied under certain conditions.



BUG #10
RTL text directionality ignored

If you simply type some right-to-left script such as Arabic or Hebrew into the main body of a document, it is laid out in the proper display order, but if you attempt to alter the character rotation to anything other than 0° by using "Format > Character …" from the menu, and selecting "Rotation/Scaling" from the "Position" Tab, nothing happens. The text stubbornly remains aligned at 0°.

None of the "Asian" or "Complex text layout (CTL)" settings have any effect on this behavior whatever, regardless of what "language" (script) is chosen for either.

If, on the other hand, you create a frame, and then enter some left-to-right text such as English, the entire string may be selected and set to either 90° or 270°. This works correctly, of course, but if you then copy a segment of correctly displayed right-to-left text, and paste it into the new frame, the inserted characters are set to the desired 90° or 270° orientation. So is this an acceptable work-around? No!

It is absolutely *not* a work-around, since the copied text reverts to its entry/storage order rather than its correct display order. It's as if a segment of **English language text** were pasted into a frame containing 270° character rotation, but then displayed as shown in the frame in the left margin!

txet egaugnal hsilgnE – rotated 270° but …

41 Gamma's Multi-Lingual Scholar was one U.S. example; Chulalongkorn University's CU-Writer was a Thai example.

42 The term 'not quite 26' refers to the fact that some of these were listed as 'not fully implemented' – and the first right to left language (as far as I am aware) was Arabic, although that was still listed as 'planned' at the time StarOffice 7 was released.

The Document Foundation's Mission

The TDF Manifesto[43] says 'We commit ourselves … to support the preservation of mother tongues by encouraging all peoples to *translate, document, support, and promote our office productivity tools* in their mother tongue.' Although this sounds more like a request for marketing support, the spirit seems clear.

On the 'Impeccable documents in just a few clicks' section on the 'Discover Writer' page,[44] we find the statement 'If you need to *use different languages* in your document, Writer can handle that, too.'

Although '*use different languages*' could be interpreted in a number of ways, Writer can't actually handle more than one language simultaneously very well at all. As the earlier examples show, even the handling of scripts leaves a lot to be desired. As for languages: two at a time, perhaps.

The Chicken and the Egg

The first two letters of the Thai alphabet that every school child learns are Gaw (chicken) and Kaw (egg), as shown in the sample page to the right. Since we've already introduced a little Thai in this document, this is a perfect segue into yet another comment that appeared in the '*Voices of Reason*'[45] discussion.

Figure 9: The Thai ABCs

> Is addressing these issues "too much for a bug that affects such a limited number of users....especially when there is an easy workaround (enable CTL in the options)"? Joel Madero; 18 June 2012. It depends on the TDF 'mission,' doesn't it?

It seems snippy to point out that enabling CTL as a 'workaround' cuases as many difficulties as it addresses, so I won't, but:

On 12 July 2012, the poster who described the forum participants using the term '*Voices of Reason*' disputed the 'limited number of users' assertion, and looking at the decided lack of activity in the Document Foundation Mail Archive's Language Subcategories, it's difficult not to wonder if perhaps there is a chicken and egg situation in play. Those who find Writer confounding but don't care (or even know how) to unravel all its options simply don't use it. While I regularly use Thai and Greek in various writings for instance, I am unacquainted with any Thai speaker who uses LibreOffice, which I find curious. Thais, after all, have been FOSS advocates for years.

The fundamental requirement of a word processor, remember, is to get characters and words on paper (or whatever medium you choose) correctly and efficiently. Features such as style management, table and frame creation, generation of indexes and tables of contents, and so forth turn out to be of minimal importance if the characters you type can't be correctly and efficiently – and easily – handled.

If this document comes across as critical, rather than an attempt to promote improvements in LibreOffice, consider how a potential user with a need for mixing scripts or languages in a single document might react when evaluating Writer for use in a corporate, academic or governmental setting.

When using AbiWord for instance, which has nowhere near the feature set of LibreOffice Writer, none of the text layout exercises presented in the first thirteen pages of this document present any difficulties. Like LibreOffice, it fails to justify the last line of right-to-left text properly but offers a decidedly simpler

43 See https://www.documentfoundation.org/foundation/

44 See https://www.libreoffice.org/discover/writer/

45 The the '*Voices of Reason*' discussion is described in the 'Bug Report as Posted' section beginning on page 47.

method (using Format > Paragraph > clicking the 'Right to Left Dominant' Check Box) of permitting a user to control this. Recall from Figure 8 that Writer can't do this.

This ends the Bug Reproduction section of the document, but a few more sections are included as support material, including:

- CTL Confusion: … in which I attempt to justify my comment in the opening preface that 'the entire subject area of Languages, Scripts, and Text Layout ('complex' or otherwise) is a mystery to many.' This begins on page 20.
- Is Rework Justified? – Or even Practical?: … page 21.
- Script-dependent Font Replacement: … This hypothetical modification to enhance Writer's handling of multiple scripts begins on page 28.
- Is Linux Font Substitution Part of the Problem? … in which I explain my suspicion that the behavior I characterized as Annoyance #1 may not be entirely the fault of LibreOffice and that the unix/linux fontconfig routines may be contributors. This begins on page 31.
- Languages, Scripts, Alphabets, & Locales: … a brief and somewhat imprecise overview of the distinctions between these and other terms. This begins on page 33.
- Observing Modifier Key Behavior: … more detailed examples of the modifier key behavior introduced in *Key Codes & Scan Codes*. This begins on page 45.
- Bug Report as Posted: … in order that this document stand on its own, it seemed useful to include the contents of the bug report titled 'Some Language & Script Handling Bugs and Reviving a Dead Horse' that was submitted to The Document Foundation's Bugzilla site. This final section begins on page 47.

We'll begin with the CTL Confusion section.

CTL CONFUSION

The term 'complex' – as in 'complex text layout' – is relative. If all one knows is addition, then subtraction might seem 'complex' and long division a nightmare. To be a mathematician, however, a student needs to continue learning algebra, geometry, calculus and so forth. Likewise, a word processor with aspirations of universality must at least transparently support the world's most common scripts and languages.

Unless a user (or developer) happens to regularly work with multiple scripts (not just multiple languages) in a single document, there is every likelihood that much of this subject area, to say nothing of terms like Kashideh or Reverse Boustrophedon, would seem complex. Unsurprisingly, there aren't many such users!

The LibreOffice guides are as good as or better than user documentation for any software I've used in the past[46] but even they are fairly muted on the meaning and intended uses of 'Complex Text Layout.'

Page 17 of the most recent iteration of the LibreOffice Getting Started Guide (for version 4.2) says:

> "The LibreOffice user interface is available in over 40 languages and the LibreOffice project provides spelling, hyphenation, and thesaurus dictionaries in over 70 languages and dialects. LibreOffice also provides support for both Complex Text Layout (CTL) and Right to Left (RTL) layout languages (such as Urdu, Hebrew, and Arabic)."

Page 67 of the LibreOffice Writer Guide (also version 4.2) adds only slightly more information:

> "… select the options to enable support for Asian languages (Chinese, Japanese, Korean) and support for *CTL (complex text layout) languages such as Hindi, Thai, Hebrew, and Arabic*. If you choose either of these … you will see some extra pages under Language Settings … These pages (Searching in Japanese,[47] Asian Layout, and Complex Text Layout) are not discussed here." (Italics are mine)

Although that really should say 'complex text layout *scripts* such as *Devanagari*, Thai, Hebrew, and Arabic,' it still doesn't suggest what makes these particular scripts 'complex' or why they're 'different.'

On June 18, 2012, while speaking of CTL in the '*VOICES OF REASON*' discussions alluded to elsewhere, Joel Madero – who is certainly well acquainted with LibreOffice – made the following comment.

> "I have never used it [CTL] before so I can't even say I knew anything that it was used for until this thread, shows my ignorance."

While trolling earlier bugs for enlightenment, I encountered #60533 (9 Feb 2013), which discussed paired elements such as parentheses and brackets being 'inverted'[48] in right-to-left languages. Not one of the commentators seemed to realize that the behavior described was exactly what one should expect given the user actions described, and why it makes perfect sense in certain right-to-left languages.

An explanation of this behavior, as well as a further overview of what are supposedly CTL characteristics, with more DIY (do-it-yourself) examples using Latin keystrokes as done earlier in this document, is provided in 'Exploring_CTL.pdf.'

My categorization of CTL on the first page as a 'mystery to many' was in no way at all intended as derogatory and, hopefully, this page explains my meaning. Given the variety of life forms on our planet, and their differing adaptations, internationalization is more involved and arcane than translating strings – and that activity certainly presents enough complex issues of its own.

---

46 The first computer I used was in 1967. Although I'm a slow learner, I have encountered a lot of software in the interim.

47 This means, for example, that if a user chooses Korean as the 'Asian Language,' special options for searching Japanese text will then become available. This can certainly qualify as 'confusing,' if the user actually notices it.

48 This term is very misleading since the examples make it certain that the author meant 'reversed' or 'flipped horizontally.'

Is Rework Justified? – Or even Practical?

Justification

In *The Document Foundation's Mission* on page 17, I sense (correctly, I hope) an objective of universal utility. This seems to be confirmed by TDF's claim to provide 'equal support for all **international scripts** - Arabic, Chinese, Cyrillic, Hindi, Japanese, Korean, Latin, etc'[49] (TDF's emphasis, not mine).

In *The Chicken and the Egg* (see page 18), I expressed an opinion that the fundamental requirement of a word processor is to arrange characters on paper correctly and efficiently. In earlier parts of this paper, I believe I've documented a sufficient number of functional and logical impediments to meeting that requirement. I would argue therefore that rework is not only justified, but overdue.

Practicality

But is a rework practical? Time, cost, and other disruptions aside, consider what might be required just from those who would contemplate taking on such a project. Arguably, an ideal developer will …

- have experience with relevant programming languages and libraries;
- have knowledge of the LibreOffice architecture, code base and internal dependencies;
- be familiar with the use of a variety of input methods, operating systems, the interactions between them, and the interactions between the input methods and user applications;
- understand the considerations needed when dealing with Unicode and UTF-8 representations;
- certainly have lots of available and uninterrupted time (having a developer with corporate sponsorship would certainly be helpful; continuity through any project is always desirable);
- have a decided interest[50] in such a project, since a serious commitment will be needed;
- be literate not only in multiple human languages (not that uncommon), but in at least two human languages that utilize different scripts/alphabets (far less common);
- be at least superficially aware of aesthetic layout considerations, the art of typography and such;
- be conversant not only with terms like justification and kerning, and the differences among pictograms, ideograms, abjads and alphabets, but with more obscure terms like boustrophedon and kashideh layout;
- have not only a suitable computer with an ergonomically correct and comfortable chair, but a sturdy drum set (there will likely be much stress to bang away in any project this fundamental);

I wouldn't think there are a lot of folks who meet even these ten criteria,[51] but the developer must also:

- be patient, remaining unflappable when dealing with those committed to the status quo;
- have a willingness to put on hip boots and wade into what I suspect is some 'interesting' code. I'll define 'interesting code' shortly, since this categorization will suggest other characteristics.

Finally, of course, there are the myriad issues and side effects resulting from what will probably be

49 See http://www.opendocumentformat.org/features/

50 Or, perhaps, a desire to make a name for themselves in the coding community and gain the awe and admiration of their peers – just a thought on possible approaches to motivation.

51 Sophie Gautier summarized these more succinctly in her 4 June 2015 comments on TDF Bug 91781: 'please before making changes to a functionality make sure that you have the whole knowledge of what it does.' (That's what I meant!)

significant changes in the user interface – not that usability wouldn't be improved and even made simpler, as I expect – but that it would simply be *different*. In his comments on 6 June 2015 regarding Bug 91781, Joel Madero decried the same thing: 'There have been complaints by users and developers that things "change just for change sake".' I'm likely not alone in having been thoroughly annoyed by the interface changes in MS-Word over the years, so I absolutely agree. (old folks like me don't like change anyway!)

Significant changes, even if they are simplifications, could result in much work for those creating or maintaining macros and extensions as well as painful surgery on User Guides and other training material. Following the comment above, Joel went on to say '... changes are made without thinking of the consequences for other teams (including documentation) ...' with which I'm certainly very sympathetic.

Before presenting another reason why some argue that the behaviors I described as illogical are somehow sacrosanct, let me define the term 'interesting' as I used it.[52]

What is 'Interesting' Code?

Despite having reached the 'old retired curmudgeon' status that I have today, memories of *those* pieces of convoluted legacy code that no one quite understood, and whose original authors had long since passed on to greener pastures, still cause bad dreams. Developers generally avoid such code because they have some innate survival instinct. Like the portions of LibreOffice discussed in this document, such sections of 'interesting' code (often compared to quicksand) all seem to exhibit similar common symptoms:

- A confusing interface (hopefully that's been established); related functions are scattered across a variety of different menus or configuration files;
- Fairly intense complaints from users who (in this example) need to work with multiple languages; trolling the Bug listings reveals that, while these are not frequent,[53] they are certainly spirited;
- Lack of any useful documentation relating to the features implemented by the code;
- References to such features are characterized by a lack of precision; terms with different meanings are often used interchangeably – 'language' and 'script' are one such pair mentioned in this paper.
- A low likelihood that many otherwise qualified developers are familiar with the purpose and use of the features implemented, leading to the rationale that 'it seems to work, so I'm sure not going to mess with it.'

Since all these symptoms seem to be in evidence, my guess is that the section(s) of LibreOffice code dealing with 'Languages, Scripts, and Text Layout' form an example of such 'interesting' code. This, strangely enough, might suggest that there is hope. But first, let's look at one last argument against rework.

Impediments due to ODF Standards?

The idea of *a* standard open document format – as opposed to *the* 'Open Document Format' – is obviously a desirable goal. The industry has settled on ODF 1.2 and this standard, therefore, should not be taken lightly, and certainly should not be ignored.

Various comments in the '*Voices of Reason*' discussions seem to believe that the ODF categorizations of 'Complex' and 'Asian' are therefore fixed and mandatory (or 'sacrosanct' as I said above), i.e. categories that must be slavishly honored if the goals of document interoperability are to be achieved. For example:

In various discussions, Joel Madero blamed Microsoft for the odd taxonomy of 'Western, Asian, and

52 Briefly, it's the same meaning found in the underhanded Chinese curse 'May you live in interesting times!'

53 … the earlier Chicken and Egg comments on page 17 may partially explain this lack of frequency.

Complex Text Layout,' as did Eike, who called it 'something that MS bestowed us and even persists in file formats.'[54] Similarly, in comments under Bug 42123, Caolán McNamara said 'You're basically stuck with the ODF standard with its three categories.'

The Reality?

Some reading and experimentation suggest that these three category distinctions are incidental, and don't really constitute 'requirements' at all.

To begin with, *at least as I read it*, what the ODF 1.2 actually defines are a number of 'standard' styles to consistently specify the bizarre 'Asian' and 'Complex Text Layout' categorizations, and all these references seem to be in forms such as '`20.248 style:country-asian`' and '`20.249 style:country-complex`.' There doesn't appear to be any mandate that use of such styles is *required* for document interchange. These references seem to have been incorporated to 'officially permit' the use of such styles – perhaps for backward compatibility with proprietary formats/schemes that turned out to have been ill-conceived.

One obvious limitation is that there seems to be *only one* acceptable style for defining each 'Asian' characteristic and *only one* acceptable style for defining each 'Complex' characteristic. Since, as I hope has been demonstrated by now, neither of these is suitable or helpful for general use, it seems that the most appropriate action would be to ignore these styles – just don't specify them, and certainly don't use them.

That such an approach is possible can be demonstrated fairly easily. Using a plain text editor (*not* Writer), create a sample file exactly as shown in *Minimal Open Document File for Testing*[55] on page 42. Save the file as plain text if other formats are available, but give the file an '.fodt' extension.

You will observe that text in the three alternative scripts is displayed perfectly well:

- The short Thai segment นี้ คือ has its vowel and tone diacritic symbols correctly positioned;
- The first two characters of the Devanagari script, हि, are displayed in their proper order even though, as can be seen in the *Hindi (Devanagari u+0900-097F) Text Sample* hex dumps on page 44, those characters are stored on disk correctly in their entry order (ह followed by ि);
- The Hebrew characters שפת עברית are properly displayed in right-to-left order even though, as can be seen in the *Hebrew (u+0590-05FF) Text Sample* hex dumps on page 44, those characters are stored sequentially on disk and in memory correctly in their entry order;

Most importantly, before opening the file in any other application, set the file's attributes to 'read-only,' since Writer as well as many other applications will 'helpfully' alter the file dramatically once it has been loaded.

Once the sample file has been created and saved, open it with LibreOffice Writer by double-clicking on it if your system has assigned the '.fodt' extension's default application[56] or by explicitly opening it in some other manner. Examining the text in Writer, it becomes apparent that Writer has used all of its default settings to format the text in these three different scripts, but has done so in the odd and inconsistent manner described in the earlier sections of this document. Being able to add styles to the text in such a document is, of course, one of the benefits or using an application suite such as LibreOffice but, in this case, it seems to me that Writer has already corrupted the text by applying Asian and Complex indicators.

Without making any changes, save the document under another name, but *still in the uncompressed* '.fodt'

54 This can be found at http://lists.freedesktop.org/archives/libreoffice/2012-June/033630.html

55 Copying the text from your pdf reader's screen is recommended, as manually typing this could be fairly tedious otherwise.

56 … as I assume would be the case for most regular LibreOffice users.

format, so that you don't lose your initial creation. Opening the new document in a plain text editor will show that not only has Writer added new sections, including …

> <office:meta>, <office:settings> and its subsections including at least one 'Asian' reference, <office:scripts>, <office:font-face-decls>, <office:styles> (containing many arbitrary, undefined and – to some – even offending 'asian' and 'complex' variations), <office:automatic-styles> and <office:master-styles>.

… which, in and of itself is to be expected,[57] but what is interesting is that Writer has applied its 'Standard' style to each segment of text – even while applying its CTL font selection *as direct formatting* – strongly suggesting that no particular style is in fact required for either 'complex' or 'right-to-left' text unless the user wishes to do so. This seems to be true regardless of the 'Complex text layout' setting.

On balance, it seems to me that, due to the dramatic changes in script and language handling over the past decade, and regardless of any resulting disruption, a rework of this fundamental segment of LibreOffice needs to be undertaken if Writer is intended to remain relevant in the future, particularly if it is to achieve or retain TDF's stated aims. But if we are indeed dealing with 'interesting' code, there is some good news.

Some thoughts on an Approach

Another common characteristic of 'interesting' code is that, once it has been redesigned and reworked, it generally becomes so much smaller and simpler that suspicions some functionality must have been left out can linger for some time. Such a reorganization of language and script handling will not add complexity; on the contrary, like most architectural 'code re-factoring,'[58] experience suggests it will be a simplification.

Here are some broad steps that might be appropriate:

1. Establish definitions and appropriate taxonomies. Terms such as *script*, *alphabet*, and *language* need to be clearly defined and distinguished, for example. Propositions regarding the relationships among these need to be stated and verified. Example definitions/propositions might be:
   - A script is a block of related symbols defined in the Unicode standard.
   - An alphabet is composed from character symbols defined in one script.
   - A written alphabetic language uses at least one alphabet, often supplemented by symbols (usually shared punctuation and similar) from another script.
   - Some written languages may use more than one writing system.

   See the section *Languages, Scripts, Alphabets, & Locales* on page 33 for additional thoughts.
2. Identify and state the key functions related to handling of scripts and languages within the system. Text layout has already been mentioned, but spell check, hyphenation, collation, justification and other such functionality should also be listed and defined. Many of these interact, of course, depending on the current attributes of some other characteristic such as script or language.

   Not only will grouping these appropriately help to sensibly arrange menu options (and some such efforts are apparently underway), but the converse should also be considered – menu structure proposals made from a usability standpoint can often lead to better underlying code structure.
3. Identify and state the implementation layers – the broad system components in which any of the behaviors of concern could be implemented. These might include the keyboard, the machine BIOS, the operating system, input methods, any number of parallel applications, etc.

57 Certainly all of the styles in use for the document must be documented.

58 To 're-factor code' is a common IT euphemism for correcting underlying design flaws or poor implementations without suggesting any fault with the original coders (who may often be the same coders doing the re-factoring)..

4. Establish which taxonomic categories and/or functionality should properly fall into which implementation layer; more importantly, identify which are stable and which are in transition, such as the migration of text layout functions to the operating system.[59] Examples *might* include:
    - The operating system is responsible for the machine's user interface as well as basic character entry, including input methods.
    - An application's user interface may differ from that of the operating system if a user desires.
    - Both dynamic placement/arrangement of base characters with any required diacritics as well as substitution of such combinations with composite characters depends on the particular script in use, and is handled to one degree or another by contemporary operating systems and/or their input methods. Operating system support for such functionality has been steadily improving so placement of this functionality still may be considered a moving target.
    - Certain non-essential text layout functionality, such as justification, is and will likely always be handled by those applications for which this support is relevant and expected.
    - Identification of syllables, word breaks and similar letter groups may be done by the operating system, but in some scripts and languages where reliance on a dictionary (or equivalent) is needed, must be accomplished at an application level. Such functionality is in transition.
    - Evaluation and/or correction of spelling, grammar, and context-appropriate writing style is currently handled at the application level. This class of functionality will likely remain the province of specialized applications such as (but not limited to) 'office' software.
    - … and so forth. The pairing of functions and implementation layers must be clarified if undesirable and confusing interactions are to be avoided.
5. Identify and avoid any hard coded assumptions.[60]
    - Isolate 'real' western defaults from chauvinistic ones or those based on ignorance. Understand that 'Default' doesn't imply and isn't equivalent to 'Normal.' Left-To-Right may be the 'Default' simply because English is the native language of processors and programming languages, but it is no more 'Normal' than Right-To-Left, which actually predated Left-to-Right by centuries.
    - Another example of an unwarranted assumption might be that only one secondary language and script will be used in a document, so combining the script choice and language definition as a unit in one location is an acceptable arrangement. It isn't.
    - Reject the tendency to associate the application interface language with the language(s) used in a document. While the most common use case might be that only one language will be used and that it will be the same as what is used in the interface, isn't a given.

      A good example of how such an assumed but invalid association can needlessly complicate the

59 In 'Before and After Unicode: Working with Polytonic Greek'; Donald Mastronarde opines 'In terms of ideal human interface design, the input scheme should be a service of the operating system, so that it is available in any application in which the user may wish to enter the specialized characters.' My contention is that LibreOffice Writer is already a beneficiary of the transition the author envisioned, but continues to unnecessarily duplicate much of this functionality.

60 My go-to example of developers permitting unrecognized assumptions to creep into a design would be an inexperienced database administrator who declares a U.S. postal code column as containing five numeric characters. Although this is as minimal a constraint as one could apply (ergo, already inept database design), the developer has also unknowingly precluded adding postal codes (and therefore customer addresses) for neighboring Canada, whose format is different. This is doubly frustrating when much better methods are not only simple to implement and more general, but result in better integrity constraints. All assumptions need to be identified, and it is an important responsibility of a designer to always consider the *general* truths behind any *specific* facts – e.g. it isn't *generally* true that postal codes have identical formats.

functionality of any software (not only Writer) is described toward the conclusion of the section titled *Is Linux Font Substitution Part of the Problem?*.

- There is some fixed relationship between the number of stored symbols or characters and the number that are displayed or printed. There isn't.
- There is some relationship between the order in which symbols or characters are stored and the order in which they are displayed or printed. There isn't.

Generalize! … and treat as many settings as possible as equally valid variables. Designs must make as few specific assumptions as possible, and therefore be as free of 'exception' handling (at least regarding functionality) as possible.

6. Overall views of the scope of any development project should exist prior to any coding beyond proof-of-concept implementations. Like the unrecognized assumptions described in footnote 60, lack of an overall scope definition often results in implementations that inadvertently preclude enhancement or expansion, particularly when adapting to unforeseen future needs. In some circles, this is known as protecting against the *Law of Unintended Consequences*.
7. Good design is characterized by many factors, but the two most important here are:
   - there is no need for any implementation to cover all possible planned functionality for work to proceed, but …
   - … any interim work done must never be done in such a way that precludes the ability to extend the existing system or add further functionality. Re-read footnote 60.

A variety of informed opinions should be solicited to compensate for the absence of any specialties listed under the *Practicality* paragraph on page 21. These should include details of the definitions, functions, and categorizations listed above, as well as identification of unrealistic assumptions, as contributions to any ongoing reviews of the project definition and scope. Opinions should include those of:

- … at least one Computer Science practitioner from each of several countries whose languages represent the majority of scripts in use. It would certainly be appropriate to have at least one regular user of scripts such as Arabic, Cyrillic, Devanagari, Greek, Hebrew, at least one Japanese script, and Thai.
- … at least one Computer Science professor who is a native speaker/writer in each major non-Latin script. Universities in India, Thailand, Japan, Russia, the UAE, and Israel (to name a few) all have well-respected CS programs from which to draw recruits (and perhaps some formal student participation). University involvement, as Apple has shown in the past, can have a far reaching influence on generational choices for software and hardware.
- … interested authors, journalists, and office workers – including from government entities who may have recently adopted LibreOffice – who utilize languages with these scripts.
- … as many scholars of dead languages from anywhere, particularly those academics who require, or feel they are more productive with, specialized software to adequately meet their needs.

Review existing Bug Reports and enhancement requests for other text layout issues that may be related to, or possibly caused by, the behaviors described in this document. Examples such as those below[61] abound and, although possibly unrelated to the behaviors identified in this document, their descriptions might contribute to understanding requirements for redesigned code or more detailed documentation:

61 This paper deals primarily with Writer, but see Bug 55793 (NEW since 9 Oct 2012), for example, dealing with Impress.

- Bug 38159 (NEW since 6 Oct 2011): 'Better full text justification with auto character scaling and paragraph level adjustment';
- Bug 60533 (NEW since 9 Feb 2013): This report, related to swapped parentheses and similar paired punctuation, was mentioned earlier in *CTL Confusion* (page 20), and is an example of an area where additional documentation might be helpful;
- Bug 62846 (NEW since 28 Mar 2013): This reports incorrect Unicode mappings within Writer;
- Bug 76014 (NEW since 11 Mar 2014): 'Writer: Hyphenation dashes seem misplaced dependent on default printer on LO launch';
- Bug 82374 (NEW since 8 Aug 2014): 'Autohyphenation at 11pt in all fonts leaves no space between last character and hyphen.'

Handling of text layout, script management, and language functionality in LibreOffice can be both simplified and improved with thoughtful planning and the contributions and collaboration of the larger user community.

Incremental Improvements

Avoiding incremental improvements until comprehensive designs have been completed is, however, a practice that has fallen out of favor[62] over the past few generations. Having said that, it is possible that several of the issues introduced earlier in this paper can likely be addressed – so long as any relevant assumptions are exposed – without waiting for any architectural review of the core functionality. Right?

Well, maybe. Although perhaps illustrating the folly of simply sticking a toe in water known to be infested with alligators, one example of such a targeted improvement is given in the following section.

62 Luckily, the astronauts had already returned safely from the moon before such heresies took hold.

SCRIPT-DEPENDENT FONT REPLACEMENT

Suppose that we decide to address Annoyance #2a – the current limitation on the use of more than one so-called CTL script in a single document (see page 4) – without having to resort to the tedious style configurations described in 'Magic Workarounds' on page 12. Further, suppose that we wish to treat this as a simple 'fix' or 'enhancement' rather than waiting for a full analysis and redesign.

We might consider an enhancement that permits the user to to add definitions for handling any number of Font-Script pairs, such as those listed for the single 'Basic Fonts (CTL)' panel illustrated in Figure 4. In his 28 June 2012 comment in the '*VOICES OF REASON*' discussions about supporting such an option, Stefan Knorr suggested that we could '... end up with different pickers for every language – which hopefully no one wants.' It's hard not to agree with such a sentiment but there are, I suspect, ways to avoid that.

In order to work with use cases already described, we'll begin with entering the Hebrew text as described in Step 7 on page 2. As we observed by the end of Step 10 on page 3, and documented in Conclusion #1 on the following page, the 'Complex Text Layout' functionality will first need to be removed or disabled to even consider supporting multiple scripts. Simply excising suspicious code blocks might cause major arterial bleeding unless some heroic analysis was performed, so we assume that it will be safer to simply remove any calls to that subsystem, while leaving those that deal specifically with Language in place.

Whether and how to alter the Writer dialogs shown in Figures 3, 4, 5, and 6 is left as a thought exercise!

With CTL calls disabled so that Writer can no longer 'help' us, the arbitrary and sometimes mysterious font substitutions described in Annoyance #4 on page 8 should no longer occur. If, as seems to be the case with modern operating systems, when alternative glyphs for those not present in the user's default font are required, substitutes will be provided.[63]

■ Complex text layout (CTL): GONE !!

Assume once again that the default font in use is FreeSerif. Writer's Styles and Formatting dialog gives us a reasonably intuitive (or at least consistent) place to add the capability for specifying any desired script-specific font substitution on an as-required basis.

In the mockup to the right, a sixth icon has been added to the Paragraph, Character, Frame, Page, and List icons now standard. In order to reduce clutter and confusion on the part of users who have no need for such font substitution, this new icon[64] and the capabilities it represents will likely only be visible if some other 'advanced' setting has been chosen, but at this point we don't care.

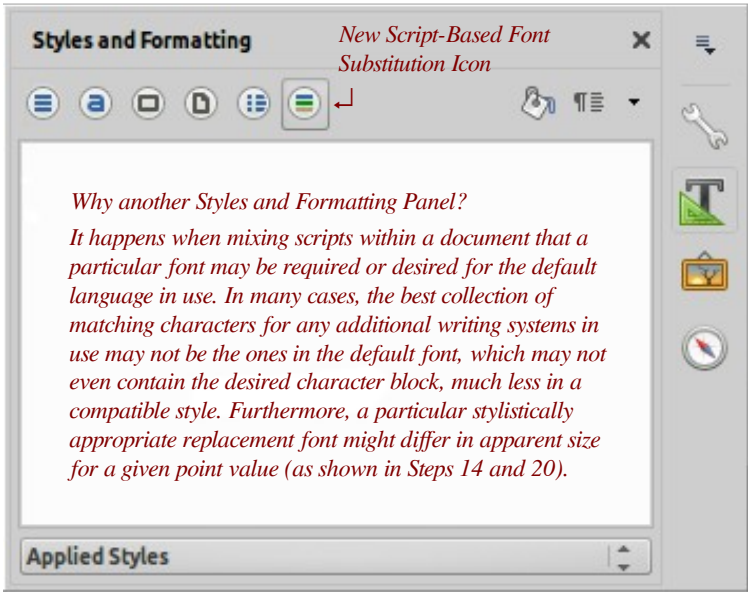


Figure 10 – Sample Approach to Script-Based Font Substitution

For our Hebrew example, the new script-based font substitution capability will not be

63 Having chosen to forgo serious analysis, some rather comprehensive testing to confirm this must, of course, take place.

64 Design of an appropriate icon, as always, presents its own issues and requires some specialized expertise. Superimposing a universally recognized non-Latin character like the Greek Ω might 'work' but would more likely be interpreted as some mathematical style choice; the common lack of distinction between 'font' and 'script,' might also need to be considered.

required and the panel will remain blank. Disconnecting the CTL capabilities should insure that, if we type either Hebrew or Thai characters, both of which are present in the FreeSerif font, no substitution will take place. Testing should confirm this to ensure that the CTL cancer has not metastasized.

If we then decide to use the 'official' Thai Sarabun font for our Thai text, and wish to adjust the 12 point Sarabun font to match the sizing of our default 12 point FreeSerif font as described in Steps 14 and 20, we will add the size adjustment more or less the same way it was done in Step 20, but with the new Script-Based Font Substitution panel. To do this, place the cursor somewhere in the Thai script segment.

Starting with an empty panel, we'll assume that the software is helpful enough to suggest that we would like to add a new entry to specify that instances of the FreeSerif font should be replaced with a different font when Thai script is encountered. The dialog might open looking something like Figure 11. If we exit the dialog without taking any further action, the contents will disappear.

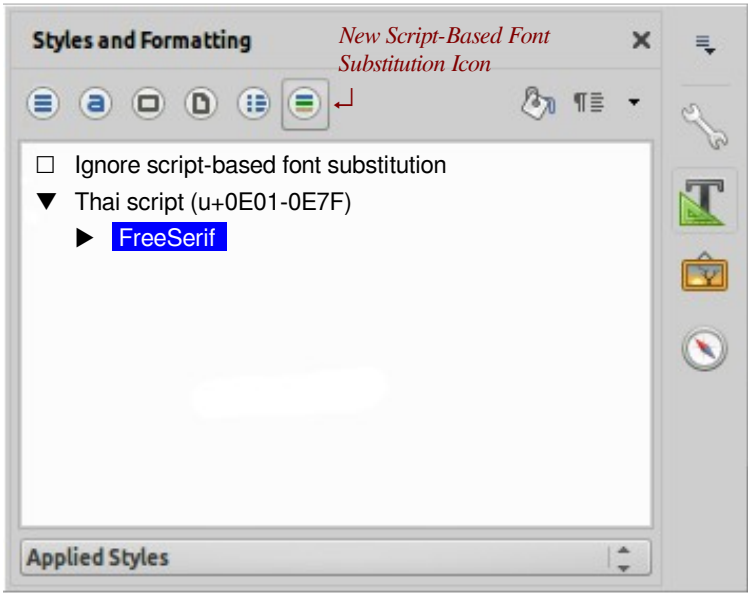


Figure 11 – Showing options for Script-based Font Substitution

Note that these settings are in no way associated with the Thai *language*, but merely the Thai *script*. Like many scripts, of course, the Thai script is used for more than one language, but for the moment we'll defer the potential implications of that. Now we have the opportunity to specify the details of our substitution. As in Step 20, we'll base these on using Sarabun at 15 point to match the FreeSerif at 12 point, but make it slightly more flexible.

In order to configure the font substitution, the FreeSerif entry under 'Thai script' will need to be opened in any usual fashion.

A panel such as that shown in Figure 12 on the right will appear. Superficially, it will operate similarly to the current 'Basic Fonts (CTL)' Panel shown in Figure 4.

To mitigate Annoyance #4 on page 8, some differences are apparent. The intended default behavior is that, if the Default Size is given as a percentage, the remaining four sub-selections (Heading, List, Caption, and Index) do not need to be set individually, but will adjust proportionally to the Default.[65]

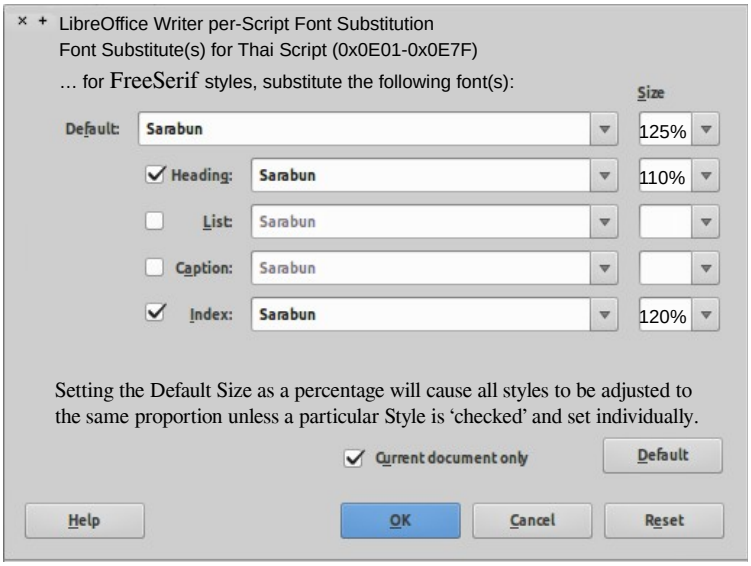


Figure 12 – Defining substitution for a Script-Font Pair

In this example, our size ratio was 15/12, or 125%. If we wish to alter specific styles, one of the appropriate boxes can be checked. In the case of the

65 These changes might first be attempted on the existing CTL panel (see Figure 4) to work out any implementation details.

Heading, for example, we might wish to have a slightly smaller size increase, so the Heading box is checked and the Size set instead to 110%, as shown here. Similarly, the Index size was set to 120%.

If the Default Size is given in points, however, all of the sub-selections will be automatically checked (or the check boxes simply would not be displayed), and the Font and Font Size for each Style must be set individually in the same manner now done in the 'Basic Fonts (CTL)' Panel.

The 'Default' button has been retained, although it isn't clear what purpose it really serves even in its current incarnation, since the default CTL settings are quite arbitrary.

When the font substitution definition panel is closed, and assuming it hasn't simply been canceled, the new Script-Based Font Substitution panel of the Styles and Formatting dialog will have changed slightly – a notation indicating the default substitute font name has been added for user convenience.

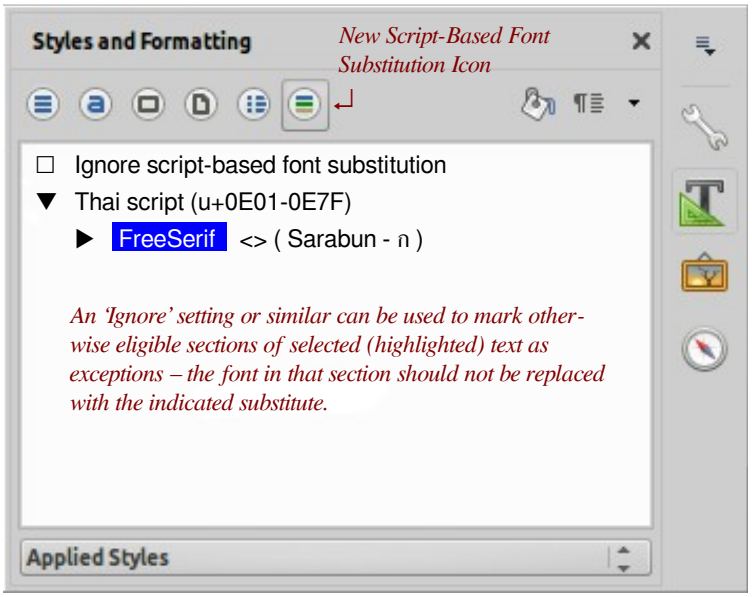


Figure 13 – Active Options for Script-based Font Substitution

The ability to utilize multiple non-default scripts is rather important in a Word Processor with aspirations to universality. How appropriate the interface described here might be would require some input from those involved in documentation and training. The term 'script,' for instance, although correct, might not mean anything to an average user. Those who might use multiple scripts in a document would, I suspect, be more likely to understand this additional complexity, so perhaps this isn't an issue.

Language sensitivity could be added if this is desired, but the general approach shown in Figure 12 will be left for others to pursue. Such language selections, however, must be secondary to Script selections.

The possible approach proposed here deals only with control of font replacements for Unicode segments that differ from the defaults for the document. These settings should not and must not overlap with other style groupings. Removal of CTL settings from paragraph and character style dialogs would also be required by this approach.

This scheme would correct Bug #8 as well as mitigate Annoyances #2+, 4, 6, and 7, but would likely still be disruptive.

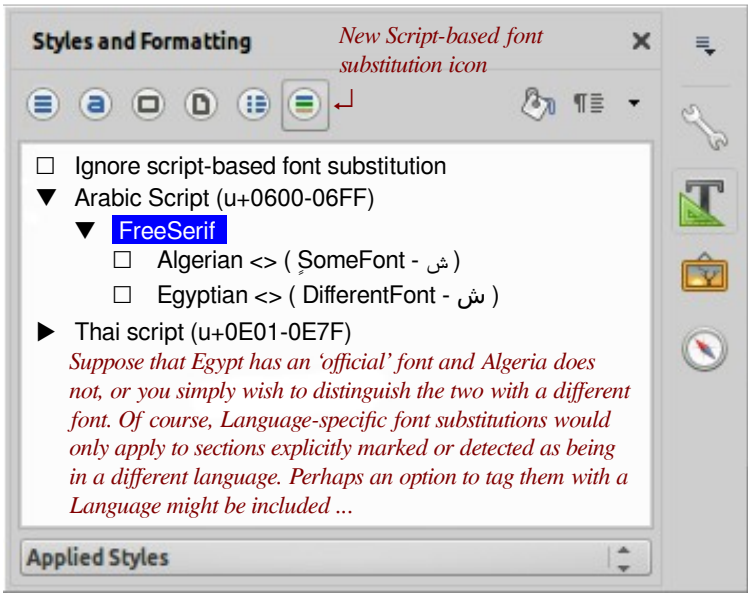


Figure 12 – Language and Script-based Font Substitution

Is Linux Font Substitution Part of the Problem?

Step 10 above identified bugs numbered 2, 3, and 4 that deal with some unnecessary and even perverse font substitutions. I suspect, but can't confirm, that this may be at least partially due to a failure of the Linux fontconfig utility to keep up with the emergence of open source fonts mentioned in 'Momentary Digression #1 for some background' on page 4. A very limited examination of such open source fonts (i.e. my own collection) suggests that a large number of these have no Foundry attribute listed at all.

The LibreOffice Localization Guide/Advanced Source Code Modifications[66] has the following to say:

> "Under unix fontconfig is used for font fallback, the original font name is sent to fontconfig and fontconfig determines the best replacement font to use depending on locale and other information."
>
> "Under unix fontconfig is used for glyph fallback, the original font name and missing glyphs are sent to fontconfig and fontconfig determines the best replacement font to use for those glyphs."

Strictly speaking, GNU/Linux – 'gnu' of course meaning 'GNU's Not Unix' – isn't Unix, but some experimentation suggests that these statements are also true of Linux. If that's the case, and if the statements from the Localization Guide remain true, my inference is that font and glyph substitutions are handled differently in Linux (and presumably in Apple's OS-X) than in Windows. The same guide also contains the following text which, although not directly relevant, is certainly cause for concern:

> "All the [*presumably alphabetic*] scripts that have at this point been defined in Unicode have been classified in OpenOffice as Latin, CTL (indic, left-to-right) or Asian (Chinese, Japanese Korean)."

This is, as many have pointed out, an arbitrary and completely absurd taxonomy of scripts. Not as absurd as Matthias de L'Obel's 1570 plant taxonomy, but just as mistaken and invalid.[67] Ignoring that for the moment, though, examine what the Linux man pages have to say about using fontconfig:[68]

The FONT PROPERTIES section begins with:

> "While font patterns may contain essentially any properties, there are some well known properties with associated types. Fontconfig uses some of these properties for font matching and font completion."

The first through third, twelfth through fourteenth, thirty-second, and thirty-third properties listed in this section are:

```
 1)  family      String   Font family names
 2)  familylang  String   Languages corresponding to each family
 3)  style       String   Font style. Overrides weight and slant
12)  pixelsize   Double   Pixel size
13)  spacing     Int      Proportional, dual-width, monospace or charcell
14)  foundry     String   Font foundry name
32)  charset     CharSet  Unicode chars encoded by the font
33)  lang        String   List of RFC-3066-style[69] languages this font supports
```

66 See https://wiki.documentfoundation.org/LibreOffice_Localization_Guide/Advanced_Source_Code_Modifications

67 In his milestone 'Stirpium adversaria nova,' publication that year, this famous botanist classified plants according to the characteristics of their leaves. The number of leaf segments of course turned out to have no general biological significance. This is a direct parallel to the taxonomic grouping of 'Latin, CTL, and Asian' that currently infests much software.

68 See http://www.freedesktop.org/software/fontconfig/fontconfig-user.html or type 'man fonts-conf' at a command prompt if you're not familiar with this facility.

69 This Internet Engineering Task Force RFC-3066 document can be seen at https://www.ietf.org/rfc/rfc3066.txt and generally specifies reliance on the ISO 639 (Languages), 15924 (Scripts) and 3166 (Countries) families of standards.

The subsequent FONT MATCHING section says, in part,

> "The canonical font pattern is finally matched against all available fonts. The distance from the pattern to the font is measured for each of several properties: foundry, charset, family, lang, spacing, pixelsize, style, slant, weight, antialias, rasterizer and outline. This list is in priority order -- results of comparing earlier elements of this list weigh more heavily than later elements."

The phrase 'this list' in the last sentence seems as if it could be interpreted in two different ways. Does "this list" refer to the list in the previous sentence or to the list in the earlier FONT PROPERTIES section? Is Foundry, for instance, the highest priority, or the fourteenth? In spite of trolling the web, it isn't clear to me, although I suspect – since it specifically says 'several properties:' – a colon followed by a list – that Foundry may actually be the highest priority. If this is, in fact, the correct interpretation, this suggests that FONTCONFIG is operating at a distinct disadvantage when open source fonts are utilized.

If Style is actually the third most important characteristic used in matching, this is also problematic when matching symbols from one Script with those of a font containing another Script. The FONTCONFIG documentation is rather vague about what exactly constitutes a 'style,' but many features considered to be font styles in western scripts have no meaning whatever in other scripts. Consider the following examples:

A	A	A	A
U+0041	U+0041	U+0041	U+0041
A	A	A	A

Identical Latin character in different 'Serif Styles'

ก	ภ	ถ	ฦ	ฤ
U+0E01	U+0E20	U+0E16	U+0E26	U+0E24
d [70]	4	5	?	A

Completely different and unrelated characters in the Thai alphabet

In the table above, four different serif 'styles' are shown on the left, ranging from a sans-serif to a heavy slab serif; these are all the same *character symbol*. Five different Thai *characters* are shown on the right, suggesting why the concept of a 'serif style' has no meaning in Thai. What look like decorative elements on each of these characters are integral parts that differentiate the character symbols, not character styles.

Like other Scripts for which the serif concept is meaningless, there are many styles used that have little applicability to Latin script. Or Cyrillic. Or Arabic. And so forth. Thus, locating Thai characters in an alternate font when the default font in use doesn't contain the Thai Unicode Script Block is a complex undertaking that depends quite a bit on aesthetic judgments; this is yet another reason why having something like the *Script-dependent Font Replacement* presented on page 28 would be quite useful.

All this also exacerbates the annoyance factor of Bugs 3, 4, and 5. If a user's default font was selected to achieve stylistically consistent symbols for any scripts required, is it any wonder that computer monitors become subject to damage from flying shoes when the user experiences unrequested substitutions?

A further issue with understanding how font and/or glyph replacement functions relates to a discrepancy between the LibreOffice Localization Guide statement that says 'fontconfig determines the best substitute font to use depending on locale … ,' and the FONTCONFIG documentation itself, which makes no mention of locale as a factor whatsoever that I could locate. Locale should *not* be a factor in script replacement.

This section explains my suspicion that reliance on FONTCONFIG may be one reason why Writer's CTL management seems to interfere with some text layout functionality that Linux handles well on its own.

---

70 As with earlier examples, these keystrokes are provided to permit easy testing with a Latin keyboard.

LANGUAGES, SCRIPTS, ALPHABETS, & LOCALES

This section could have been titled 'Languages, Dialects & Countries, Writing Systems, Scripts, Abjads, Abugidas & Alphabets, Key Codes & Scan Codes, Character Codes and Character Cells, Locales, Typefaces, Glyphs & Fonts' but that would have been unwieldy. But an awareness that such categories exist, and an ability to distinguish among them, is helpful for anyone hoping to address the issues discussed in this paper. It must be stated that although there is 'expert' support for each of the descriptions and opinions expressed in this section, contrary 'expert' opinions can just as easily be found. These notes are intended only as a starting point for your own explorations of any segment you feel may be relevant.

Languages, Dialects & Countries

A Language is a method of human communication consisting of meaningful sounds or symbols that are more or less structured by a generally accepted if not always formalized grammar. A Language may be used in aural (e.g. spoken), visual (e.g. printed or displayed), or tactile (e.g. Braille) fashions, alone or in a variety of combinations. Some ancient languages – ones that specialists have discovered how to read but not pronounce – have been unspoken for perhaps thousands of years, but are still written. Any language is presented using at least one Writing System, and several languages can be and are written with several.

The International Standards Organization has produced a set of documents – imaginatively named ISO 630-1 through ISO 630-5[71] that establish two and three Latin alphabetic codes for identifying the world's Languages. This seems like a great idea, except that a) these standards deal only with Living Languages, and b) there has never been any consensus on the differences between Dialects and Languages (e.g. when does a dialect qualify as a separate language?). The only generally agreed-upon taxonomy seems to show 'Language Families' as the universe of discourse – with each Language Family containing multiple child Languages – and each child Language possibly containing a variety of Dialects. These categorizations, as might be expected for living languages, cannot be considered static.

Fortunately, for the language-related functionality specific to office applications, such as spell checking, thesaurus use, and grammar checking,[72] the definition of Language – whether living or dead, real or artificial – can often be inferred if need be from the existence of a dictionary or similar support file(s).

Country is often a rather transient concept; many 'countries' have existed for thousands of years, but their borders, languages, cultures, scripts and even ethnic compositions may have changed dramatically over time due to political shuffles. Country names, though, are commonly used as a convenience to distinguish among what are really broad dialects of common languages, such as the sixteen 'English (*xxx*)' Languages presented as options in LibreOffice.[73]

Writing Systems

Although this term is commonly used and understood, the word 'writing' suggests that it refers literally to something 'written' or 'displayed,' but in reality the term also encompasses all the visual and tactile Language representations discussed earlier.

It is generally supposed that Writing Systems began as collections of simplified drawings of objects and, later, of concepts and/or sounds. The first symbol in Figure 13, for instance, represented 'an ox' while a

71 Until 2014, there was an ISO 639-6, but that was dropped; apparently, even for bureaucrats, five was considered plenty.

72 Soundex Codes, specific to English names (although parallels exist for other languages), also fall into this category.

73 Standardized country codes include ISO 3166-1, -2, and -3. Another source for related data can be located at the United States CIA's site https://www.cia.gov/library/publications/the-world-factbook/appendix/appendix-d.html.

logograph depicting two or three oxen represented the concept of a 'herd' rather than a specific quantity.

Eventually these symbols also began to be used and understood in context simply as phonemes (syllable sounds) used to form longer spoken words. Alternatively, symbols such as 'Sun' and 'Flower' might be merged to suggest the concept of a sunflower. Today we might call this 'scope creep' but such increasingly arcane and obscure symbol combinations enhanced the job security of the few scribes who were able to keep up with them, so this type of writing system continued to be used.

Eventually, though, the benefits of using symbols to represent more discrete sounds began to drive the development of what we know today as Alphabets;[74] this transitional process is illustrated in Figure 13.

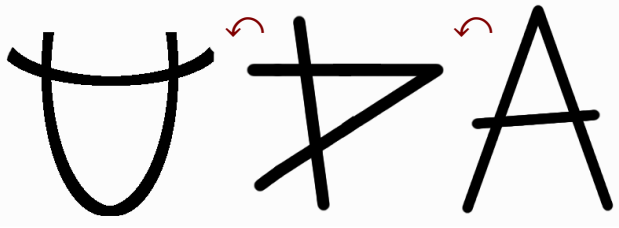


Figure 13 – from icon to syllable to letter sound

The early Phoenician icon for an Alep (Ox) is shown on the far left. Eventually, the symbol came to represent the Alep *sound* as well, rotating a little over -90° in the process, and shown in the middle symbol. After some further rotation it became what we would now consider a 'letter' – the rightmost 'A' symbol – that survives in a variety of present-day alphabets. Although the physical similarities aren't immediately evident, the name 'Alep' survives in character names such as Hebrew's *Aleph* (א) and Greek's *Alpha* (α).

All these 'systems' of writing remain in use today, not only in academic studies of ancient texts, but by any potential users of word processors. Chinese logographic characters (known as hànzì) dating from at least the 5th century are still in use, and were adopted by the Japanese – by whom they are known as Kanji. Japanese is also written using two sets of phonetic symbols known as Hiragana (sounds used in native Japanese words) and Katakana (sounds used in words borrowed from other languages).

Additional special-use writing systems have been designed for special needs; examples include Braille, the International Phonetic Alphabet system and various clerical, legal, and medical shorthand notations.

Figure 14 – the Phaestos Disk

Each writing system has a number of characteristics, although only one – directionality – is relevant to this paper. Symbols in a given writing system can be laid out in the familiar left-to-right/top-to-bottom and right-to-left/top-to-bottom directions[75]; Other layout patterns include top-to-bottom/left-to-right, top-to-bottom/right-to-left, bottom-to-top in either direction, several boustrophedon hybrids (alternating left-to-right/right-to-left patterns that get their name from the path taken by the aforementioned ox as a field was plowed), and even spiral patterns such as that shown in Figure 14.[76]

More detailed discussions of the world's variety of Writing Systems can be located easily, but the important things to note here are:

- A Language may use one or more Writing Systems.
- A Writing System may be used by many Languages.
- Writing Systems are not necessarily alphabetic.

74 With fewer symbols required to represent speech, the relative ease of learning Alphabets encouraged a rise in literacy.

75 Known rather bizarrely by much software – including LibreOffice – as 'normal' and 'RTL' respectively.

76 The two top-to-bottom layouts are referred to inadequately as 'Asian,' while the others don't seem to be supported at all.

And remember, we haven't even considered sequences of drum beats, Morse Code, hole patterns on cards and paper tape, little-endian and big-endian bit sequences in a variety of lengths both fixed and variable, and the tactile Braille. Assuming the availability of appropriate printing and presentation/editing devices, there is no reason why such tactile writing systems should not be supported by a word processor. Support for boustrophedon or spiral layouts, however, can probably be left to graphics software such as Draw.

Scripts

Some sources make no distinction between Writing Systems and Scripts. For what it's worth, that seems to me to be an untenable position, particularly when organizing software functionality.

In order to standardize the identification and transmission of symbols used to represent the world's writing systems, the Unicode Consortium took on the task of identifying each and every symbol ever used for that purpose, and giving each of them a unique number. Even more helpfully, they began organizing these symbols into 252[77] related groups called Blocks or Scripts such as the alphabetic symbols we've already seen in this document: Hebrew U+0590-05FF, Devanagari U+0900-097F, and Thai U+0E00-0E7F, for example. Examples of symbols from other writing systems include the IPA[78] 0250-02AF, Braille U+2800-28FF, Egyptian Hieroglyphics U+13000-1342F, and even Shorthand U+1BC00-1BC9F.[79]

Some Unicode blocks contain symbols that are intended as supplements for specific scripts, while others contain symbols that are not part of any particular alphabet. Such blocks may contain mathematical symbols, various types of diacritic and other marks, composite characters such as ligatures, and so forth.

These Unicode blocks are further grouped into larger sections such as European Scripts, East Asian Scripts etc. but this is *solely for convenience* since many writing systems and languages utilize symbols from more than one of these symbol groupings.[80] Although any of the character Scripts/Blocks may sometimes be equivalent to a particular alphabet, they should not be viewed as alphabets themselves. The distinction will hopefully be made clear in the next section.

Some Scripts have obvious commonalities with others, but this cannot be taken to mean there is any relationship at all between the languages that use those scripts. Cyrillic script, for instance, is based on the Greek, but with some additional symbols added for sounds the Greeks didn't use. The Greek language is written with the Greek script, and the Russian language is usually written with the derivative Cyrillic script. Linguistically, however, the Greek and Russian languages have nothing in common.

Many assume that because Thai script is primarily used by the Thai language, the Thai language is always written with the Thai script. Rendering Thai with Cyrillic script could therefore seem rather odd, but consider the Russian traveler preparing for a visit to Bangkok who might purchase a book of useful Thai phrases. Because most non-Thais (including software developers) are likely not familiar with Thai script, nor have time to learn it, phrases in a Russian guidebook are often rendered – however imprecisely – in Cyrillic script. Even more interesting, in guidebooks for an Egyptian traveler, the Thai phrase would be presented with Arabic characters running right-to-left, even though Thai is written left-to-right.

Furthermore, many languages – and therefore scripts – share common separators (such as the space), punctuation marks, tabs, paired delimiters (such as parentheses, brackets and the like) and numeric

---

77 … as of Unicode Version 7.0

78 The International Phonetic Alphabet, which isn't associated with any particular Language.

79 Sign languages are not yet part of Unicode Version 7.0, but are proposed for the 1D800-1DAAF script block.

80 See http://www.unicode.org/charts/ for listings. Language particularities of each script are provided on the Script's page.

characters[81] from the Latin block formerly known as 'lower ASCII.'[82] There are also blocks containing various types of unique punctuation, as well as a variety of graphical symbols.[83] There is, therefore, no one-to-one correspondence between Scripts and Languages.

Abjads, Abugidas & Alphabets

Most western Writing Systems such as Latin are based on the use of what are generically called 'alphabets,' but there are potentially relevant, if not important, distinctions within even that category.

The earliest form of alphabet – and still in use today – indicated only the sounds we now call consonants. Such a form is generally known by the Arabic name Abjad. If an abjad were used to write English, we might see something like 'BJD' ('abjad') or 'LFBT' ('alphabet') where the reader would need to fill in some appropriate vowel sound simply to be able to pronounce the words at all.

Later, realizing that identical words with differing vowel sounds, such as 'TRBL' in the sentence 'TH CLRNT PLR HD **TRBL** GTNG CLR **TRBL** FRM TH RSTY NSTRMNT.'[84] might be subject to misinterpretation by a reader, certain markings began to be added over, under, or inside consonants to associate them with specific vowel sounds. These slightly more sophisticated form of alphabets – still not giving equal status to vowels – are now known rather harshly as 'impure abjads' and almost exclusively use vowel symbols that are much smaller than those of the consonants. Most abjads in use today, including Arabic, Hebrew, and Aramaic, are thus 'impure.' Even the word Abjad (أبجد) is itself impure! Such ironies may explain why people just use the term 'alphabet.'

Several modern abjads add full-sized vowel characters to supplement their diacritic vowels. The presumed further decrease in purity of these abjads doesn't appear to be discussed in the relevant literature. Both types of vowels can be seen in the Thai sample given in Momentary Digression #1 on page 4.

An Abugida, another early alphabet class, began with languages having similar sounding 'filler' vowels; as the need arose to indicate multiple vowel sounds, this class – rather than adding separate diacritics to the consonants – made slight modifications to the consonants themselves which typically included attaching hooks or other appendages. In some cases, consonants were rotated to indicate different vowel sounds.[85]

Democracy, along with equal rights and status for vowels, seems to have first been introduced by the Greeks, thus making classical Greek the first true alphabet in the sense we commonly think of today.

Such an Alphabet is a set of character-class symbols from a single Script that are used by a particular Language. Each Alphabet will have a defined order, so that lists may be sorted, dictionaries easily queried, and so forth. An Alphabet may not include all symbols from its Script, and one alphabet's prescribed ordering for one Language may not be the same as that of another Language using the same Script.

An example may help clarify the distinctions between and relationships across Languages, Writing Systems, Scripts, and Alphabets:

The Cyrillic Script, originally derived from the Greek for recording the Russian language, was itself

81 … sometimes referred to as 'numerals' and/or incorrectly as 'numbers.'

82 See Bug #1 on page 2 for the effects of not accounting for this symbol sharing across Script blocks. Handling such shared symbols isn't always as straightforward as it might seem, but it does have a certain underlying logic.

83 Standard Script Codes are defined by ISO 15294 (see http://unicode.org/iso15924/iso15924-codes.html); more details about Unicode Scripts are available at https://en.m.wikipedia.org/wiki/Script_(Unicode).

84 Easy: 'The clarinet player had ***trouble*** getting clear ***treble*** from the rusty instrument.' Note that programmers may already be familiar with several Abjad examples without realizing it; 'CLR' for example is used in several scripts and languages.

85 Really! Several writing systems and alphabets for Canadian aboriginal languages are constructed this way.

eventually adopted as the basis of alphabets used in a variety of neighboring languages.[86] Surprisingly to many, it is also used in alphabets developed for some Eskimo-Aleut dialects languages spoken in the United States. Generally, alphabets are associated as much with Languages as with Scripts, suggesting some relevance to word processing applications.

Although the Latin 'A,' the Greek 'Alpha,' and the Cyrillic 'Ah' – first letters in their respective alphabets – might appear to be 'shareable,'[87] and may all use what look like (and what may even be) identical glyph representations (A, A, & A respectively in FreeSerif[88]), they are separate and distinct *characters*! Other characters that appear identical, such as the Latin 'N' (U+004E) and the Greek 'N' (U+039D) represent a consonant and vowel respectively. While certain classes of symbols may be shared across Languages, alphabetical characters are never shared across Scripts.

Perspectives on Progress

Before continuing this section, an unfair look at the changes in technology over the past several thousand years is in order. Unlike the current incarnation of LibreOffice Writer, early versions of Libre-Gouger such as that shown to the right had no restrictions on the variety of Scripts that could be mixed on a single stone[89] although, to be fair, the editing capabilities (e.g. copy, cut, and paste)) of those early versions were minimal. My apologies; I couldn't resist.

Figure 15 – Early Word Processor circa 3000 BCE
(a heavier Bold mallet was an available option)

If this discussion leaves the impression Writing Systems progressed steadily from pictures and icons through syllable sounds, characters, and finally to alphabets, it must be pointed out that recent Unicode symbol additions include a bikini (U+1F459), high-heeled shoe (U+1F460), and computer mouse (1F5AF), none of which, as near as I've been able to determine, were used by the ancient Egyptians or Greeks.

Key Codes & Scan Codes

The 'process' of word processing usually begins by entering symbols with a keyboard and, since comments in the '*Voices of Reason*' discussions[90] suggest that examination of keyboard behavior might be relevant to addressing the issues discussed in this paper. It isn't, but it still seems useful to make a few comments.

English-speaking users likely refer to the keys of a computer keyboard with names like 'the A key' or 'the Tab key.' Thai-speaking users might actually use those same terms, but are just as likely to call the identical key 'ปุ่ม ฟ' ('the ฟ key' or more literally ' the ฟ button').

For the most part, despite the accoutrements of some specialized products, keyboards have no concept of

86 In addition to Russian, these include Belarusian, Bosnian, Bulgarian, Karelian, Kildin Sámi, Komi-Permyak, Kurdish, Macedonian, two Mari, one Mongolian, Montenegrin, Ossetian, some Romani, Rusyn, one Serbian, Tajik, and Ukrainian alphabets. Although similar, a variety of characteristics, e.g. alphabet composition and sorting order can be seen.

87 If you care, their distinct Unicode character points are u+0041 (65d), u+0391 (913d), u+0410 (1040d) respectively.

88 But – as displayed here in the FreeSerif font, you many notice that the Latin A is slightly taller than the other two.

89 The famous Rosetta Stone of 196 BCE, containing Hieroglyphic, Demotic, and Greek Scripts, is a good example of this.

90 In his 10 February 2014 comment under Bug 47969 (NEW since 27 Mar 2012), Joel said '… detecting keyboard layouts is currently not part of the code if I'm not mistaken …' Whether detecting these layouts might be useful or required for other purposes is an open question, but it doesn't seem relevant to CTL discussions.

Letters or Symbols; the key names simply reflect whatever is printed on the tops of the keys. Keyboards are, at their core, a collection of anywhere from 40 to well over 100 switches – usually arranged in rows and staggered columns, and internally identified by their particular row-column intersections. Groupings of keys – often indicated by different coloring – are devoted to particular uses, such as alphanumeric, punctuation, cursor movement, editing, and so forth.

On my venerable Northgate OmniKey Ultra, the key marked with the 'A' operates switch #38.[91] When that switch is pressed, the keyboard sends the eight bit sequence 00011100 (decimal number 28 or the hexadecimal value `0x1c`) to the computer. When the key is released, the sequence 10011100 (156; `0x9c`) is sent. Such sequences originating within the keyboard are called Scan Codes.

Note that any key-down sequence differs from its corresponding key-up sequence in the first bit, meaning the key-up value is always equal to the key-down value plus 128; this assists the computer in keeping track of key presses that might not appear in sequence. When key #38 is released, and barring any other information being available, the computer uses its knowledge of the keyboard to interpret the 'key #38 down + up' sequence as the single byte 01100001 (97 `0x61`), the ASCII and Unicode value of the lower case 'a,' not the capital 'A.' Even with 'ฬ' or 'ฦ'[92] printed on the key, a Thai keyboard works the same.

So, yes, at this stage, the computer knows only about symbols represented by numbers between zero and 127 – what used to be called the lower ASCII range and now Unicode's Basic Latin range. At this low level, all computers recognize only the characters that English uses, since users and their languages are really not of much interest. Command lines, batch files and shell scripts, programming languages, and machine op codes were initially created by English speaking folks and there seem to be few if any benefits that would be worth the effort to change this. Operating systems have matured to the point, however, that any language and script can eventually be recognized without users even being aware of this limitation.

To see a simple example of how modifier keys such as Shift, Ctrl, Alt (or combinations) are used to interpret key presses differently, consider the following sequence of key presses using the Shift key:

Shift ⇓	Switch #50 closed:	Bit sequence sent from the keyboard is **0**0010010 (18; `0x12`).
		Modifier key pressed[93] and active; the computer waits to see what's next.
A ⇓	Switch #38 closed:	Bit sequence sent from the keyboard is **0**0011100 (28; `0x1c`).
		Character key pressed; but the 97 must be 'modified' (reduced by 32).
A ⇑	Switch #38 opened:	Bit sequence sent from the keyboard is **1**0011100 (156; `0x9c`).
		Character key released; interpret the 'a' as 'A' (65) due to modifier.
		Modified value is forwarded for further interpretation or processing.
Shift ⇑	Switch #50 opened:	Bit sequence sent from the keyboard is **1**0010010 (92; `0x146`).
		Modifier key released; modification buffering is stopped.

Not all modified key sequences are sent on for further processing; the Ctrl+Alt+Del sequence made infamous by DOS and Windows is acted upon immediately by the system itself. More detailed examples of how key presses are modified to reflect the user's intentions, as well as a means for observing these in action, are provided in the *Observing Modifier Key Behavior* that begins on page 45.

---

91 The internal scan codes used in your keyboard may be entirely different.

92 Thai has 44 consonants along with a variety of vowels and tone markings, but has no concept of capital and small letters; these two characters are different and unrelated, with the less frequently used ฦ character entered by using the Shift key.

93 In this example, the left shift key is shown, but the results using the right shift key would be the same. What is important to note is, at a low level, each switch on a keyboard is independent of any other, and can be detected separately if desired.

Character Codes and Character Cells

A Character Code is, as described earlier, the specific number assigned to represent a particular symbol; although 'character' is often used colloquially as a synonym for 'letter,' that is limiting and thus inaccurate.

A Character Cell is what appears on paper as a single character position; the distinction can be seen in the earlier example from Step 24: the Thai snippet นี้ คือ appears to (and does) occupy four Character Cells.

As we saw, however, it is made up of seven distinct Characters, each of which has its own symbol. The initial consonant has both vowel and tone mark diacritics, followed by a space, another consonant with a vowel diacritic, and a final, seventh character symbol. The importance of recognizing this distinction cannot be overstated, and failure to do so can cause unexpected and potentially confusing results.

In the Thai example above, the `m17n` library on my system has used its default font layout table for Thai Script (`THAI-GENERIC.flt`)[94] to reorganize the consonant-vowel-tone sequence into the composite glyph that is ***displayed*** in the Character Cell. In this example, storage of the separate symbols isn't affected.

On most systems, there is a variety of ways to enter characters or symbols that aren't given their own dedicated keys. Operating systems and applications each provide utilities such as Writer's 'Insert > Special Character' menu option. Operating systems permit users to designate a 'Compose Key,' while applications such as Writer have Auto-Correct options. In the former case, typing R-Alt, (, c[95] results in the symbol © being displayed; with Writer's default auto-correct replacement table, typing (C) accomplishes the same thing. What is different from the Thai example immediately above is that, in each of these cases, the single © (`U+00A9`) symbol replaces not only what is displayed, but what is stored in memory and on disk.

So, why do we care? In this example, we don't, and the single © is probably the result we want. But consider some other examples: type the word 'aesthetic' and note that it contains nine characters. Type the word 'waffle' and note that it contains six characters. Use the right and left arrow keys to move through the characters you typed. Then, using any of the methods above, or other 'helper' utilities, type these words as 'æsthetic' and 'waffle'[96] in order to mimic a certain level of typographic sophistication. Once again, using the right and left arrow keys, confirm that the words now consist of only eight and four 'characters' respectively. Determining whether to store or merely display such composites can be a kerfuffle! (Sorry.) To observe a few more examples, copy the following line of characters into a Writer document:

→ – aesthetic – æsthetic – waffle – waffle – นี้ คือ – ½ – ¾ – ü – ™ – kerfuffle

The → arrow, like the © symbol, is both stored and displayed as a single character. The word aesthetic is not only stored and displayed identically, but passes an English spell-check. The version spelled with the æ character symbol (`U+00E6`),[97] also stored and displayed the same way, looks nice, but fails a spell check.

The version of 'waffle' with the ffl ligature (`U+FB04`),[98] although stored and displayed differently than the

---

94 See http://manpages.ubuntu.com/manpages/precise/man5/mdbFLT.5.html for a short description. On Ubuntu, these font layout table files are located in /usr/share/m17n/. Other systems are in use but script-specific font layout tables, which use something akin to regular expressions with glyph positioning as well as substitution variables, seem to give the best results, particularly since font-specific layout tables can be created where required to support decorative fonts.

95 This assumes that the Right Alt key has been defined as the Compose key; your own setup might be entirely different.

96 The 'a' and 'e' has been replaced with 'æ' (`U+00E6`) and the 'f,' 'f,' 'l' sequence has been replaced with the 'ffl' ligature (`U+FB04`). See the interesting Q&A on the http://unicode.org/faq/ligature_digraph.html page.

97 This is stored in UTF-8 representation as the two byte sequence 0xC3A6; using the conversion for two byte UTF symbols described in 'Hebrew (u+0590-05FF) Text Sample' on page 44, the actual Unicode `U+00E6` identifier can be extracted.

98 This is stored as the three byte UTF-8 sequence 0xEFAC84; the conversion for three byte UTF symbols is described in

version without the ligature, still passes a spell check. Is 'æ' a victim of ligature discrimination? Sort of.

Some apparently 'composite' characters such as the Spanish ñ qualify as full-fledged alphabetic characters because they are listed in a country's alphabet. Others, such as the 'æ' in our sample, are viewed as alphabetic characters in some countries (in this case, Denmark, Iceland, and Norway), but as ligatures in others. Some ligatures are recognized by some dictionaries,[99] and some are automatically 'expanded' before being passed to spell checking routines. In short, there is still a wild-west quality in this area.

Ignoring the contentious history that determined which composite characters qualify for their own Unicode assignment,[100] make several copies of the text line. Now, using the Alt key with the arrow keys, observe that three arrow presses are needed to move through the single displayed น้ำ Character Cell.

Thus, 'composite' characters that are only used for display can still be edited, although Writer's quirky cursor positioning during this process requires strict attention. Using the copies of the sample line, experiment with the ← Backspace and Delete keys as well to observe how these behave.

Now type some right-to-left text; the Hebrew example from Step 7 (a p , space g f r h ,) will suffice. This should be displayed as שפת עברית. Once again, copy this to several lines for some experimentation.

In much documentation, the ← Backspace key's function is described as 'delete the character to the left,' a description reinforced by the left arrow printed on that key on most keyboards. Even the official name for the corresponding Unicode symbol ⌫ (U+232B) is 'Erase to the left.' The function of the Delete key is similarly described as 'delete the character to the right,' and again reinforced by the Unicode name for the ⌦ symbol (U+2326), which is 'Delete/Erase to the right.'

The actions you observe as the ← Backspace and Delete keys are used on the right-to-left Hebrew sample suggest an interesting source of confusion, since the terms 'right' and 'left' are clearly based on an assumption that text is always laid out in a left-to-right sequence. Such names are not likely to change, and it is also unlikely that the direction of the arrow on the backspace key will ever be dynamic. Most keyboard layouts also subtly imply that, rather than being functionally related to the ← Backspace key, the Delete key is functionally related to the Insert key, since they are generally placed side by side.

Documentation and training efforts for many cultures therefore need to exercise care when associating these key labels with their actions, e.g. describing the ← and → keys as 'previous' and 'next.'

Luckily, LibreOffice seems to handle itself well when dealing with all the differences between characters and character cell contents. Given the power of the Auto-Correct functions, however, perhaps users should be required to answer a few security questions before being allowed to add to the AutoCorrect replacement table to insure their own safety.[101]

Locales

Under 'Languages,' LibreOffice Help says that Locale '… influences settings for numbering, currency and units of measure,' and provides a user with 'a user interface (UI) of [your] chosen language.' But the help text, as written, confirms the idea that Writer is designed such that a user may have one and only one spell check function for each of the 'Western,' 'Asian,' and 'CTL' languages.'

---

both the 'Thai (u+0E01-0E7F) Text Sample' and 'Hindi (Devanagari u+0900-097F) Text Sample' beginning on page 43.

99 … in some cases, as alternate spellings, such as encyclopaedia (a & e) or encyclopædia (the æ ligature) for encyclopedia.

100 … which isn't really relevant, since Writer must deal with what is, but researching this can generate some laughs.

101 This suggestion is tongue-in-cheek, of course, but thoughtless entries can cause surprisingly inappropriate results.

The Locale also specifies the default Language, Country, Encoding (e.g. UTF-8), default sorting orders (Collation), address formatting (placement of postal codes), currency, currency formatting, number formatting (e.g. comma versus period as decimal point), and default paper sizes. In LibreOffice, many of these are handled more flexibly in Calc than in Writer, but that is outside the scope of this paper.

A principle to guide any work in this area is that neither an operating system nor application locale setting should preclude entry and presentation of text in any script for which an appropriate font is available.

Typefaces, Glyphs & Fonts

These terms are used rather indiscriminately, regardless of the sources consulted and, like other terms discussed in this section, aren't necessarily relevant to the larger thrust of this paper; nonetheless, since they are found throughout, it seemed appropriate to at least comment on them briefly to avoid any misunderstanding.[102] I'll reiterate that these definitions are not intended to be technically complete, and are therefore not necessarily accurate.

A Typeface is a coordinated set of stylistically consistent designs for at least one alphabet and, most likely a supplemental array of ancillary symbols intended for use with whatever language might use that alphabet. Typefaces intended for the same alphabet will contain the same alphabetic characters and symbols, but the remainder of the symbols may not be the same for all typefaces.

A Glyph is a single unique drawing or other realization of one symbol in a typeface. Such symbols are not necessarily equivalent to complete characters; some characters and character cells may be formed from more than one glyph.

A Font was originally a collection of glyphs in a specific size for a particular typeface, but that distinction has become less and less relevant as computer technology and scalable typefaces became available. A Font now would be more usefully described as a set of mathematical instructions with which a computer can transfer the design of each glyph to some display device.[103] A key difference in the current definition of the term font is that many, if not most, currently used fonts contain alphabets and/or symbol groups from more than one of the Unicode blocks or Scripts described earlier.

A Font Family is a related group of fonts that are stylistically matched, but in different weights or styles; examples would be Bold or Italic versions. While these are separate font files, most applications list only the standard version and use the others transparently when called for.

102 For those with an interest, one source that I do consider both authoritative and entertaining is "The Elements of Typographic Style" by Robert Bringhurst; ISBN: 978-0-88179-212-6, now in its fourth edition.

103 Which, in this context, also includes printers and similar devices.

MINIMAL OPEN DOCUMENT FILE FOR TESTING

At the end of 'The Reality?' on page 23, creating a minimal read-only open document text file (.ODT) using a plain text editor was suggested. The contents of the figure below can be used for this purpose.

```
<?xml version="1.0" encoding="UTF-8"?>

<office:document
 xmlns:office="urn:oasis:names:tc:opendocument:xmlns:office:1.0"
 xmlns:style="urn:oasis:names:tc:opendocument:xmlns:style:1.0"
 xmlns:text="urn:oasis:names:tc:opendocument:xmlns:text:1.0"
 xmlns:table="urn:oasis:names:tc:opendocument:xmlns:table:1.0"
 xmlns:draw="urn:oasis:names:tc:opendocument:xmlns:drawing:1.0"
 xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"
 xmlns:xlink="http://www.w3.org/1999/xlink"
 xmlns:dc="http://purl.org/dc/elements/1.1/"
 xmlns:meta="urn:oasis:names:tc:opendocument:xmlns:meta:1.0"
 xmlns:number="urn:oasis:names:tc:opendocument:xmlns:datastyle:1.0"
 xmlns:svg="urn:oasis:names:tc:opendocument:xmlns:svg-compatible:1.0"
 xmlns:chart="urn:oasis:names:tc:opendocument:xmlns:chart:1.0"
 xmlns:dr3d="urn:oasis:names:tc:opendocument:xmlns:dr3d:1.0"
 xmlns:math="http://www.w3.org/1998/Math/MathML"
 xmlns:form="urn:oasis:names:tc:opendocument:xmlns:form:1.0"
 xmlns:script="urn:oasis:names:tc:opendocument:xmlns:script:1.0"
 xmlns:config="urn:oasis:names:tc:opendocument:xmlns:config:1.0"
 xmlns:ooo="http://openoffice.org/2004/office"
 xmlns:ooow="http://openoffice.org/2004/writer"
 xmlns:oooc="http://openoffice.org/2004/calc"
 xmlns:dom="http://www.w3.org/2001/xml-events"
 xmlns:xforms="http://www.w3.org/2002/xforms"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:rpt="http://openoffice.org/2005/report"
 xmlns:of="urn:oasis:names:tc:opendocument:xmlns:of:1.2"
 xmlns:xhtml="http://www.w3.org/1999/xhtml"
 xmlns:grddl="http://www.w3.org/2003/g/data-view#"
 xmlns:officeooo="http://openoffice.org/2009/office"
 xmlns:tableooo="http://openoffice.org/2009/table"
 xmlns:drawooo="http://openoffice.org/2010/draw"
 xmlns:calcext="urn:org:documentfoundation:names:experimental:calc:xmlns:calcext:1.0"
 xmlns:loext="urn:org:documentfoundation:names:experimental:office:xmlns:loext:1.0"
 xmlns:field="urn:openoffice:names:experimental:ooo-ms-interop:xmlns:field:1.0"
 xmlns:formx="urn:openoffice:names:experimental:ooxml-odf-interop:xmlns:form:1.0"
 xmlns:css3t="http://www.w3.org/TR/css3-text/"
 office:version="1.2"
 office:mimetype="application/vnd.oasis.opendocument.text">
 <office:body>
  <office:text>
   <text:p >Now is the time for all good men to abandon CTL styles.</text:p>
   <text:p/>
   <text:p >นี้ คือ</text:p>
   <text:p >हिन्दी भाषा</text:p>
   <text:p >שפת עברית</text:p>
   <text:p >This is some Hebrew text: שפת עברית and it's right-to-left.</text:p>
   <text:p/>
  </office:text>
 </office:body>
</office:document>
```

To the right is a 'hex dump' of the file with its cursor shown at location `0x094A`[104] – the start of the first section using non-Latin characters, in this case Thai.

In the three tables below, the different script segments are expanded to show each individual byte (in hexadecimal) on the first row. Their UTF-8 binary representation is shown in the second row as stored in memory and on disk.

Row three shows the portions of the UTF-8 bit stream representing the actual Unicode values, while the fourth row arranges them in their normal single or double byte notation. Row 5 shows the Unicode values represented by those bytes, while the sixth shows the Latin keys used as described earlier in this document and the non-Latin characters that result.

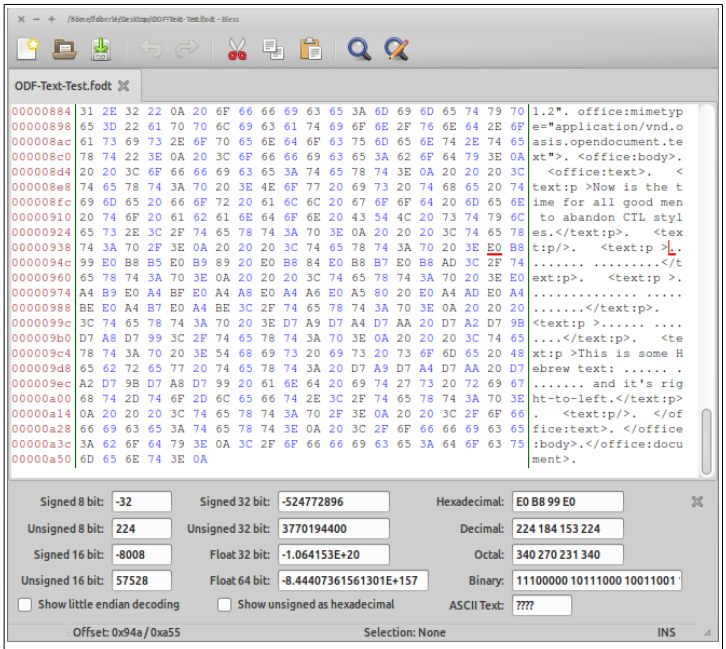


Figure 10 – Hex dump of Open Document Text Sample

Thai (u+0E01-0E7F) Text Sample

The nineteen bytes (four character cells in green) of the Thai text segment นี้ คือ beginning at `0x094A` are:

E0	B8	99	E0	B8	B5	E0	B9	89	20
11100000	10111000	10011001	11100000	10111000	10110101	11100000	10111001	10001001	00100000
0000	111000	011001	0000	111000	110101	0000	111001	001001	0100000
00001110	00011001		00001110	00110101		00001110	01001001		00100000
u+0E19 (3609)			u+0E35 (3637)			u+0E49 (3657)			u+20(32)
o = น			u = ◌ี			h = ◌้			space

E0	B8	84	E0	B8	B7	E0	B8	AD
11100000	10111000	10000100	11100000	10111000	10110111	11100000	10111000	10101101
0000	111000	000100	0000	111000	110111	0000	111000	101101
00001110	00000100		00001110	00110111		00001110	00101101	
u+0E04 (3588)			u+0e37 (3639)			u+0e2D(3629)		
8 = ค			n = ◌ั			v = อ		

In UTF-8 format, any byte beginning with a '1' is part of a multi-byte character. Two or more leading '1' bits indicate the first byte of such a group; any byte beginning with '10' is a continuation byte. In the first byte, the number of leading '1' bits indicate the total number of bytes in the character.[105]

Thus, when packaged as a UTF-8 stream in memory or on disk – shown as a hex dump in line one and as the equivalent twenty-four bits in line two – the sixteen bit Thai Unicode character น (`u+0E19`) is extracted from the UTF-8 representation into groups of 4, 6, and 6 bits as shown in the third line.

What is significant to note in this example is that the Thai base character น and its two diacritics (a vowel and a tone mark) are stored separately in memory *as they were entered*. Thus, it is only on screen or in

104 The location is based on Unix line endings and will vary depending on the end-of-line lengths of your operating system. This particular view was made using the Linux Bless utility, but any other hex viewer or editor should suffice.

105 This scheme facilitates efficient determination of cursor movements as well as deleting and backspacing, which users expect to work on character cells, as opposed to individual elements. LibreOffice uses Alt to navigate those elements.

print that we see them occupying a single character cell as the composite นั้. Thai script is therefore considered to be 'complex.' The reality, however, is that this situation is really no different than typing a French word having the letter á, except for the fact that, rather than storing the letter 'a' (u+0061) and the acute accent ´ (u+00B4) separately, they are converted on input to the single á (u+00E1) character.

Note also in this Thai sample that the fourth character cell consists of only one byte (the tenth) and is an example of how many alphabets share non-alphabetical characters with the Latin script.

Hindi (Devanagari u+0900-097F) Text Sample

The initial sixteen bytes (of twenty-eight) of the Hindi text segment हिनदी beginning at 0x0973 are:

E0	A4	B9	E0	A4	BF	E0	A4	A8
11100000	10100100	10111001	11100000	10100100	10111111	11100000	10100100	10101000
0000	100100	111001	0000	100100	111111	0000	100100	101000
00001001	00111001		00001001	00111111		00001001	00101000	
u+0939 (2361)			u+093F (2367)			u+0928 (2344)		
h = ह			i = ि			n = न		

E0	A4	A6	E0	A5	80	20	
11100000	10100100	10100110	11100000	10100101	10000000	00100000	*Sixteen Bytes displayed*
0000	100100	100110	0000	100101	000000	0100000	*in Six Character Cells*
00001001	00100110		00001001	01000000		00100000	
u+0926 (2342)			u+0940 (2368)			u+20 (32)	
d = द			I = ी			space	

The Devanagari script also uses three bytes to represent each character in UTF-8. Of interest here is that, while the stored order in memory is the same as the entry order, the rendering order of the first two characters is exchanged, in this case because the ि vowel always precedes its associated consonant.

Hebrew (u+0590-05FF) Text Sample

The initial fifteen bytes (eight character cells) of the Hebrew segment שפת עברית beginning at 0x09A5 are:

D7	A9	D7	A4	D7	AA	20	
11010111	10101001	11010111	10100100	11010111	10101010	00100000	*Fifteen Bytes displayed*
10111	101001	10111	100100	10111	101010	0100000	*in Eight Character Cells*
00000101	11101001	00000101	11100100	00000101	11101010	00100000	
u+05E9 (1513)		u+05E4 (1508)		u+05EA (1514)		u+20 (32)	
a = ש		p = פ		, = ת		space	

D7	A2	D7	9B	D7	A8	D7	99	D7	AA
11010111	10100010	11010111	10011011	11010111	10101000	11010111	10011001	11010111	10101010
10111	100010	10111	011011	10111	101000	10111	011001	10111	101010
00000101	11100010	00000101	11011011	00000101	11101000	00000101	11011001	00000101	11101010
u+05E2 (1506)		u+05DB (1499)		u+05E8 (1512)		u+05D9 (1497)		u+05EA (1514)	
g = ע		f = כ		r = ר		h = י		, =	

Although the Hebrew characters are stored sequentially as they are entered, they are displayed and printed in right-to-left order as would be expected with any Unicode blocks in the u+0590-08FF range.

The Bottom Line: with very few exceptions, although LibreOffice needs an awareness of script and language related differences, it doesn't need to care about the arbitrary and obsolete 'Asian' and 'Complex' classifications currently responsible for many of its quirks. The Input Method handled all these examples.

OBSERVING MODIFIER KEY BEHAVIOR

At the end of the *Key Codes & Scan Codes* section, the behavior of modifier keys such as the two shift keys was introduced. This section describes this behavior in a little more detail for both default key interpretations as well as for those intercepted and translated by input methods.

The outputs presented are those displayed using the Linux key press monitor utility `xev`, although there are similar utilities for any commonly used operating system. The sequences described on page 38 are repeated first to show how they are displayed using `xev`. First is the entry of the single letter a:

User Key Press Sequence	Output reported by the Linux xev utility
Press and release the A key. State 0x10 means 'unmodified.' Keycode 38 is the key identifier. Keysym 0x61 is the unmodified code used as the default interpretation of keycode 38. Unicode u+61 is "a". The KeyRelease event forwards the 'a' character (u+61) to the next step.	KeyPress event, serial 34, synthetic NO, window 0x3c00001, root 0x2c5, subw 0x0, time 174984398, (103,-9), root:(990,472), state 0x10, keycode 38 (keysym 0x61, a), same_screen YES, XLookupString gives 1 bytes: (61) "a" XmbLookupString gives 1 bytes: (61) "a" XFilterEvent returns: False KeyRelease event, serial 37, synthetic NO, window 0x3c00001, root 0x2c5, subw 0x0, time 174984541, (103,-9), root:(990,472), state 0x10, keycode 38 (keysym 0x61, a), same_screen YES, XLookupString gives 1 bytes: (61) "a" XFilterEvent returns: False

The next example shows how the shift key alters the state of the system so that key presses entered are altered to produce a character other than what an unmodified bit stream from the keyboard would normally produce. In this case, the entry of an uppercase/capital A is shown:

User Key Press Sequence	Output reported by the Linux xev utility
Press the Shift key and hold it until the next key is pressed. Note that the state for the next key press will be 0x11, 'shifted.'	KeyPress event, serial 37, synthetic NO, window 0x3c00001, root 0x2c5, subw 0x0, time 174986619, (103,-9), root:(990,472), state 0x10, keycode 50 (keysym 0xffe1, Shift_L), same_screen YES, XLookupString gives 0 bytes: XmbLookupString gives 0 bytes: XFilterEvent returns: False
Press and release the A key. With the state 'shifted,' keycode 38 is now interpreted as keysym 0x41. Unicode u+41 is "A". The KeyRelease event forwards the 'A' character (u+41) to the next step.	KeyPress event, serial 37, synthetic NO, window 0x3c00001, root 0x2c5, subw 0x0, time 174987066, (103,-9), root:(990,472), state 0x11, keycode 38 (keysym 0x41, A), same_screen YES, XLookupString gives 1 bytes: (41) "A" XmbLookupString gives 1 bytes: (41) "A" XFilterEvent returns: False KeyRelease event, serial 37, synthetic NO, window 0x3c00001, root 0x2c5, subw 0x0, time 174987295, (103,-9), root:(990,472), state 0x11, keycode 38 (keysym 0x41, A), same_screen YES, XLookupString gives 1 bytes: (41) "A" XFilterEvent returns: False
Release the Shift key. Releasing the shift key sets the state for the next key press to 0x10.	KeyRelease event, serial 37, synthetic NO, window 0x3c00001, root 0x2c5, subw 0x0, time 174987717, (103,-9), root:(990,472), state 0x11, keycode 50 (keysym 0xffe1, Shift_L), same_screen YES, XLookupString gives 0 bytes: XFilterEvent returns: False

The third example illustrates how the interpretation of key presses is altered when an Input Method (in this case, iBus) is used and the Thai TIS-820 keyboard layout is activated as described in Step 13. The

actual switch to an alternate keyboard mapping also produces output from the `xev` utility (as would be expected, since `xev` responds to mouse as well as keyboard actions), but these are ignored here. The next two examples use the same keys as before:

User Key Press Sequence	Output reported by the Linux xev utility

Press and release the A key.
State 0x10 means 'unmodified.'
Keysym `0xdbf` is the unshifted code altered by the input method to replace the default interpretation of keycode 38 with the UTF-8 sequence `e0b89f`. Extracting the Unicode value from the UTF-8 sequence gives `u+0E1F`.
The KeyRelease event forwards the character 'ฟ' (`u+0e1f`) to the next step.
	`KeyPress event, serial 43, synthetic NO, window 0x3c00001,`
	`root 0x2c5, subw 0x0, time 175013665, (840,-344), root:(1727,137),`
	`state 0x10, keycode 38 (keysym 0xdbf, Thai_fofan), same_screen YES,`
	`XLookupString gives 3 bytes: (e0 b8 9f) "ฟ"`
	`XmbLookupString gives 3 bytes: (e0 b8 9f) "ฟ"`
	`XFilterEvent returns: False`
	
	`KeyRelease event, serial 43, synthetic NO, window 0x3c00001,`
	`root 0x2c5, subw 0x0, time 175013808, (840,-344), root:(1727,137),`
	`state 0x10, keycode 38 (keysym 0xdbf, Thai_fofan), same_screen YES,`
	`XLookupString gives 3 bytes: (e0 b8 9f) "ฟ"`
	`XFilterEvent returns: False`

Thus, pressing the key marked with an A produces the Thai character ฟ.

The final example shows how the shift key produces an entirely different Thai character from the same key; as mentioned earlier, Thai, like many scripts, has no concept of "capital" letters.

User Key Press Sequence	Output reported by the Linux xev utility

Press the Shift key and hold it until the next key has been pressed.
Note that the state for the next key press will be 0x11, 'shifted.'
	`KeyPress event, serial 43, synthetic NO, window 0x3c00001,`
	`root 0x2c5, subw 0x0, time 175015953, (840,-344), root:(1727,137),`
	`state 0x10, keycode 50 (keysym 0xffe1, Shift_L), same_screen YES,`
	`XLookupString gives 0 bytes:`
	`XmbLookupString gives 0 bytes:`
	`XFilterEvent returns: False`

Press and release the A key.
Keysym `0xdc4` is the shifted code altered by the input method to replace the default interpretation of keycode 38 with the UTF-8 sequence `e0b8a4`. Extracting the Unicode value from the UTF-8 sequence gives `u+0E24`.
The KeyRelease event forwards the character 'ฤ' (`u+0E24`) to the next step.
	`KeyPress event, serial 43, synthetic NO, window 0x3c00001,`
	`root 0x2c5, subw 0x0, time 175016495, (840,-344), root:(1727,137),`
	`state 0x11, keycode 38 (keysym 0xdc4, Thai_ru), same_screen YES,`
	`XLookupString gives 3 bytes: (e0 b8 a4) "ฤ"`
	`XmbLookupString gives 3 bytes: (e0 b8 a4) "ฤ"`
	`XFilterEvent returns: False`
	`KeyRelease event, serial 43, synthetic NO, window 0x3c00001,`
	`root 0x2c5, subw 0x0, time 175016645, (840,-344), root:(1727,137),`
	`state 0x11, keycode 38 (keysym 0xdc4, Thai_ru), same_screen YES,`
	`XLookupString gives 3 bytes: (e0 b8 a4) "ฤ"`
	`XFilterEvent returns: False`

Release the Shift key.
Releasing the shift key sets the state for the next key press to 0x10.
	`KeyRelease event, serial 43, synthetic NO, window 0x3c00001,`
	`root 0x2c5, subw 0x0, time 175017291, (840,-344), root:(1727,137),`
	`state 0x11, keycode 50 (keysym 0xffe1, Shift_L), same_screen YES,`
	`XLookupString gives 0 bytes:`
	`XFilterEvent returns: False`

This time, pressing the A key along with either Shift key produces the Thai character ฤ.

BUG REPORT AS POSTED

Bug Report Title: 'Some Language & Script Handling Bugs and Reviving a Dead Horse'

This submission deliberately ignores the usual recommended practice of creating separate reports for each bug or enhancement request. This is done for two reasons:

1) It seems almost certain that many – if not all – of the bugs and/or questionable behaviors described in this report are very closely related, as are the enhancement suggestions. It seems more advantageous to view these as a forest than as individual trees (as has been done in the past with a few of them) in order to properly assess their impact on non-Latin word processing.

2) Rather than suggesting that these bugs and questionable behaviors be 'fixed,' my suggestion is rather that a complete redesign and (hopefully) overhaul of that part of LibreOffice related to Languages, Scripts, and Complex Text Layout (CTL) be undertaken, with the whole concept of CTL being abandoned completely. It is unnecessary with contemporary operating systems, and it causes problems.

The attached 'Bugs-and-a-Horse.pdf' document will provide those interested with detailed explanations, steps for reproducing the behaviors, and explicit examples demonstrating how to test these behaviors without requiring *ANY* knowledge of the particular languages or scripts (and there are several) used in the examples. I'll address the important 'Dead Horse' reference after first enumerating the Bugs.

The specific bugs and annoyances being reported here can be summarized as follows:

Bug #1: Writer considers shared Latin characters (space, punctuation, etc.) as specifically Latin when detecting non-Latin scripts, causing confusing cursor movement in right-to-left scripts as well as other inappropriate behaviors. –see page 2 of 'Bugs-and-a-Horse'.

Bug #2: Writer displays/reports an incorrect 'Text Language' in use. –see page 3 of 'Bugs-and-a-Horse'.

Bug #3: Writer makes arbitrary, unrequested, inappropriate, and often befuddling font and glyph substitutions, although its choices can usually be determined with a little effort. –see page 3.

Bug #4: Writer then makes even further font substitutions (i.e. beyond and in addition to those referenced in Bug #3), and these require quite a bit of effort to determine. –see page 3.

Bug #4 might be specific to Linux distributions. –see page 31 of 'Bugs-and-a-Horse'.

Bug #5: Writer reports the wrong Font-in-Use. –see page 3.

Annoyance **#1**: I've classified the effort required to determine the font in use as Annoyance #1.

Recommended Enhancement **#1**: Under Tools > Options > LibreOffice > Appearance, include an option to add a color setting to indicate font substitution and/or glyph substitution (per step 10, bullet three in the 'Bugs-and-a-Horse' document). A related enhancement might be an indication of whether 'bold' or 'italic' text is using a legitimate bold or italic variant of the font in use, or whether either is being simulated.

Annoyance **#2+**: Only one so-called 'CTL' script can be used in a document without a significant level of pain and confusion. –see page 4 of 'Bugs-and-a-Horse'.

The reason for the '+' is that the CTL acronym is used as if it refers to a characteristic of language, rather than as a characteristic of a script. This is not mere quibbling over semantics, as this misunderstanding adversely affects many of Writer's operations.

Annoyance **#3**: Directly shifting between different 'Tools > Options' panels isn't always possible. –see

page 8 of 'Bugs-and-a-Horse'.

Annoyance **#4**: The five language settings on the 'Basic Fonts (Western)' and 'Basic Fonts (CTL)' panels are tedious to use and could be made a bit more friendly. –see page 8 (ibid).

Unnumbered Annoyance: I occasionally need to restart Writer rather than simply close the 'Languages' panel in order for CTL font and size settings to take effect, but I have never been able to pin down the circumstances required to reproduce it, so it is only mentioned briefly –see page 9.

Annoyance **#5**: Handling of keyboard shortcuts when non-Latin Input Methods are selected could use some attention. Under some conditions, [Ctrl + whatever] isn't detected –see page 9.

Bug #6: Characters entered can be 'lost' or ignored under repeatable conditions. –see page 10 (ibid).

Recommended Enhancement **#2:** Under Tools > Options > LibreOffice > Appearance, include an option to color-code text that has direct formatting applied. (per step 25 in 'Bugs-and-a-Horse.')

Unnumbered Annoyance: When multiple cells in a Writer table are selected, the 'Clear Direct Formatting' function doesn't seem to work. I haven't thoroughly tested this, so it is only mentioned briefly.

Bug #7: Incorrect style applied when setting CTL fonts. –see page 11 of 'Bugs-and-a-Horse'.

Annoyance **#6**: Recommended workarounds for several issues discussed here are tedious –see page 12.

Annoyance **#7**: Layout of 'Languages' Panel can be misleading –see page 13 of 'Bugs-and-a-Horse'.

Bug #8: The taxonomy of the 'Languages' panel choices is logically flawed. –see page 13.

Bug #9: Full Justification is improperly completed in Right-to-Left scripts (ironic since Arabic's more sophisticated Kashideh RTL justification seems to be handled well. –see page 13.

Bug #10: Characters in Right-to-Left scripts cannot be rotated; furthermore if they are pasted into a frame formatted with character rotation, their order is reversed. –see page 17.

In order to become familiar with the scope of these bugs and annoyances, I would suggest reading through the entire document before attempting to follow all the steps in detail, but that's just my opinion.

While 'Bugs-and-a-Horse' began as a series of steps needed to reproduce the behaviors of concern, it soon grew into a rambling (and possibly opinionated) essay. The 'dead horse' reference, an allusion to the English language idiom 'beating a dead horse,' reflects another objective of 'Bugs-and-a-Horse': to reopen some occasionally passionate earlier discussions about similar bug reports that ended up being ignored, avoided, unresolved, or simply misunderstood. I'll be referring to these discussions as the '*VOICES OF REASON*' discussions – the name is based on a July 2012 comment during those exchanges where poster Shahar Or said 'the voices of reason' have been present here.' These can be seen at any of these links:

- http://lists.freedesktop.org/archives/libreoffice/2012-June/033552.html
- http://lists.freedesktop.org/archives/libreoffice/2012-July/034427.html
- http://lists.freedesktop.org/archives/libreoffice/2012-July/034958.html

The 'Bugs-and-a-Horse' document is intended to support my beliefs that a) due to progress, some fundamental sections of LibreOffice have been become obsolete and perhaps even irrelevant since they were first developed, and b) the time to revise this subsystem is now, and it should be given priority. Therefore, some historical and architectural perspective – showing why the time to revise this subsystem is now – is included.

A second document, titled 'Exploring_CTL.pdf' is also attached as a supplemental discussion of various aspects of what is known as 'Complex Text Layout.'