L2/22-028

Bidi in programming languages and markup languages

Kent Karlsson
Stockholm
2022-01-17

Individual contribution

Introduction

We will here focus on how bidi should be handled in IDE editors and to some extent compilers and interpreters that allow bidi characters (or, more precisely, strong RTL characters). Not every programmer uses an IDE, or an editor that “knows” which programming language is being edited. That case is not covered here.

In https://www.unicode.org/L2/L2022/22007-avoiding-spoof.pdf (Avoiding Source Code Spoofing), there are some ideas for how to handle bidi for programming languages, and avoid some possible spoofing, or readability, issues. Here I’ll give my take on some of the issues raised, and my recommended handling of bidi in programming language source code including script languages, but also extended to (data) markup languages (like HTML, XML, JSON) now often also edited via IDEs, particularly for XML and JSON files that are used in systems and edited “at the same time” as some program source related to such XML/JSON files and have some of the same issues as programming language sources.

First of all, there is *nothing* requiring that a programming language source code editor (typically in an IDE), use the bidi algorithm *at all*. That is probably the easiest approach to handling this particular spoofing/readability problem. Even if not using the bidi algorithm, the editor may use reverse glyphs for RTL characters, thereby enabling cursive joining and ligatures; though the RTL portions (character-wise, no bidi algorithm) will be mirrored/backwards. That may be well acceptable for string literals in *source code*. It may be less acceptable for identifiers, if one at all allows for RTL characters in identifiers, and even less for comments that are supposed to be read as natural language text. Identifiers are only visible to the programmers, who need to put up with sometimes strange syntax in the programs anyway (and they are not visible to the end-user of the program). Note also that identifiers often are really not “words” that one can find in a dictionary; they are strings of letters (and sometimes digits), portions of which often can be “words” or abbreviations of words or multiple words. The bidi algorithm is really intended for words that at least potentially could be found in a dictionary. Any use of the bidi algorithm, ligatures, or cursive joining may give strange results for pseudo-words like identifiers.

And worse, applying the bidi algorithm to “paragraphs” (read: lines of code) can foul up the entire reading of the source code line. But one could apply a very restricted form of the bidi algorithm, isolated to identifiers, each identifier bidi processed one by one; but the program *lines* be displayed LTR. See below for more details. That will still retain source readability, while allowing identifiers (that are really constructed from words one may find in a dictionary) to use RTL characters, and get each identifier rendered “normally” (i.e. not what would be backwards in the script used).

Suggested handling of invisible characters and strong RTL characters

Invisible characters

https://www.unicode.org/L2/L2022/22007-avoiding-spoof.pdf already has some suggestions with regards to invisible characters. Invisible characters include characters of general category Cc that are not recognised in the programming language in question. NLFs like LINE FEED, sometimes with CARRIAGE RETURN are usually recognised, as is CHARACTER TABULATION. The other Cc characters usually already give rise to compilation errors and also error indications in IDE editors.

But there are other invisible characters in Unicode. Of particular interest here are the bidi control characters. These should not be allowed in programming languages or markup languages, except *perhaps* in comments. For their use in string literals and attribute values, one can still use the character escape mechanism of the language in question for them not to occur in the source, but still available for use at run time (or data read time). For identifiers, there should be no need to allow them at all, even if formally allowed. A parser/compiler (and IDE editor) can have more stringent rules that what is specified in the programming language (markup language) specification. If nothing else, as programming style rules.

Handling of strong RTL characters in programming/markup languages

We here discuss explicit RTL characters, not when they are expressed as character escaped (like \u*nnnn*, &x*nnn*;).

We need to consider 3 cases:

1) Identifiers, including tag names and attribute names in markup languages like XML.
2) String literals, including attribute values in XML and similar.
3) Comments in programming languages and in XML and similar markup.

All programming languages (so far) and all markup languages (so far) are designed to be read left-to-right (LTR). So outside of the three cases above, everything is strictly LTR, under all circumstances, including the general layout of lines. Thus all “whitespace”, all numerical literals, all punctuation and all operators are all strictly strong LTR. And the paragraph direction (for bidi) is always LTR.

It may be hard to convince a general-purpose text editor of that, but an editor that “knows” which programming language (or markup language) is being edited should have no problems (in principle). They can already distinguish different kinds of identifiers (static, parameter, constant, etc.), comments, etc. So limiting any application of the bidi algorithm as described here should not be any major problem, if the implementor of the programming language adapted editor selects to implement bidi at all.

For the three cases above, the bidi algorithm may be applied, but to each identifier individually, each string literal part individually, and each line of comment individually. Thus, those are the “paragraphs” (in bidi algorithm terms) there are in a program (or markup) source file, with the paragraph direction (for the bidi algorithm) as LTR. Identifiers and string literals should not mix strong LTR and strong RTL characters in one identifier or one string literal part. If there is no strong LTR or RTL character in the identifier (like “__”) or string literal part, it should be considered strong LTR. If the identifier contains one or more strong LTR characters, the entire identifier or string literal part should be considered strong LTR. If it contains one or more strong RTL characters (but no strong LTR character) the entire identifier or string literal part should be considered strong RTL. There should be no bidi control characters (not counting those that are expressed via character escapes), or at least they should not be interpreted by the bidi algorithm when displaying the source code in a "source code aware" editor. Such characters, if present, should give error messages by the compiler/parser, and not allow further processing (parsing, compilation, data structure formation). For "interpreted" languages: Essentially the same as for compiled programming languages, but applied for "interpretation unit" (usually lines), one by one.

For comments, there should still be no bidi control characters, and the "paragraphs" (lines) for the bidi algorithm (with paragraph direction LTR) must exclude the comment start and comment end markers. In case of multiline comments, each line is bidi processed individually, and should exclude the comment continuation marker, if any. The latter refers to the common practice of having " * " at the beginning of "internal" lines of a multiline comment, like:

```
/**
 * This is a multiline comment with
 * comment continuation markers ( * ).
 *
 * The programming language syntax for comments
 * do not require any such continuation markers,
 * but they are common practice, and supported
 * by several IDEs (when generating such comments).
 */
```

A minor luxury could be to consider single line breaks to be LS and double line breaks (ignoring comment continuation markers) to be PS for the purposes of the bidi algorithm. Not sure this would make much difference, in the absence of bidi control characters.

Other editors

An issue with the approach mentioned above, is that when cutting and pasting (or writing directly) in an editor that does not "know" which programming language it is, the display of the source code lines can still get messed up in the presence of RTL characters. One cannot say "just don't do that", because there are fully legitimate reasons to do so: like writing textbooks, example collections, exercise lists, or less ambitious, notes or discussions of program snippets (including in chats).

This is unfortunate, even though one, in principle, could have a cut-and-paste system that "automatically" sprinkles (in an appropriate manner) the pasted program snippet with bidi control characters (or HTML bidi markup, if the target is HTML encoded) to get the same effective layout. Though a possibility in principle, I have no great hopes of that being implemented, let alone widely and commonly implemented. So unfortunately, one will have to do that manually, and that is then up to the author of the textbook, example collection, chat question, etc., who want to use RTL characters in their program source code.

As mentioned above, not everyone uses an IDE for writing or viewing program source code. Some "plain text" editors do have some support for some programming languages. That support may be extended to handle bidi in source code as recommended above. But for those editors that do not have any special handling of program (or data) source code, the occurrence of RTL characters, and if the editor supports bidi, will mean that source lines will be messed up to a degree that they may be misread (due to the bidi rearrangement) or look so horrible (due to the bidi rearrangement) that the author/viewer either turns to another editor, or the author eliminates the RTL characters or turns off the bidi processing as a user option (if the editor has such a user option).
