L2/12-186R

A Proposal for Bidi Isolates in Unicode

Aharon Lanin, Mark Davis, and Roozbeh Pournader
July 24, 2012

Live document: http://goo.gl/K6qtV
Document history: #heading=h.6rlabmox2h39

Abstract

HTML/CSS recently introduced bidirectional isolates to allow for much better handling of bidirectional text in HTML. Using an isolate around an opposite-direction phrase or an unknown-direction phrase prevents it from inadvertently distorting the display of its surroundings in a robust way that could not be achieved with older constructs (for more details, see below). Furthermore, an unknown-direction phrase can be displayed in the direction automatically estimated from its content independently of the direction of the paragraph in which it appears.

Currently, however, there are no corresponding features in Unicode. This presents several problems:

- The new technology is unavailable to solve the same issues arising in non-HTML documents.
- There is no way to faithfully translate the new HTML/CSS constructs into plain text on copy/paste from HTML.
- The HTML/CSS isolates have to be defined and implemented as a complicated superstructure on top of the Unicode Bidirectional Algorithm (UBA). The complexity has led to difficult-to-fix bugs in current browser implementations of isolates.

This document proposes specific changes to the UBA to support isolates. It provides a high-level summary of the proposal, a description of changes that is being proposed in parallel to the CSS working group, and then a detailed description of the exact changes entailed in the UBA specification (UAX #9).

Table of Contents

High Level Proposal

The purpose of this section is to outline the proposal conceptually. For a definitive specification of the proposal, see "Detailed Changes to UAX #9" below.

We propose extending Unicode with new codepoints and changes to the UBA that would offer two new features:

- Directional isolation
- Direction estimation for a phrase nested within a paragraph

Directional isolation means that a piece of text can be surrounded with invisible formatting characters that reduce its effect on the bidirectional ordering of its surroundings to that of a

neutral character. This is in contrast to the existing embedding formatting characters (LRE, RLE, PDF) which have the effect of a strong character on their surroundings. Otherwise, isolates are quite similar to embeddings: they declare a direction for the text inside it, and can be nested inside another isolate or embedding (and vice-versa). Isolates are just a kinder, gentler form of embedding that prevents opposite-direction phrases from scrambling their surroundings.

To see the need for isolates, take the case of a Hebrew article title, "מאמר מעניין", followed in our English document by a hyphen and the article's date of publication, let's say "14 July 2012". When no formatting characters are used, the result is a mess:

> 14 - מאמר מעניין July 2012

That's because the UBA has no way of knowing whether the "14" is a part of the RTL phrase or of the LTR surroundings, and happens to guess incorrectly that it is a part of the RTL phrase.

Unfortunately, and here we get to the real point, surrounding the Hebrew title with explicit embedding formatting characters (RLE and PDF) does not change this display at all, since the explicit embedding still affects its surrounding the same as an RTL character.

In contrast, surrounding the Hebrew title with the new isolate formatting characters will result in the intended ordering:

> מאמר מעניין - 14 July 2012

(Since isolates are obviously not available yet, we achieved it here via traditional means discussed in the "Isolates vs Embeddings and Marks" section below.)

Direction estimation for a nested phrase means that a part of a paragraph can be explicitly marked to be of an unknown direction, which is to be determined from its content using the same first-strong algorithm already specified by the UBA for whole paragraphs. It makes perfect sense to limit this capability to a kind of isolate. Isolates that specify an explicit direction would still be used where the direction of their content is known (and easily accessible) to the document author, but if that direction is not readily known, it can be specified to be guessed automatically. This is a very common scenario when generating documents on the basis of templates and data.

Specifically, we propose four new Unicode formatting code points:

- LRI (LEFT-TO-RIGHT ISOLATE): marks the beginning of a left-to-right isolate.
- RLI (RIGHT-TO-LEFT ISOLATE): marks the beginning of a right-to-left isolate.
- FSI (FIRST-STRONG ISOLATE): marks the beginning of a first-strong isolate, i.e. one whose direction is determined by applying rules P2 and P3 to the isolate's content as if it were a separate paragraph.
- PDI (POP DIRECTIONAL ISOLATE): marks the end of an isolate.

The actual locations to be used for these codepoints, of course, are up to the UTC, but U+2066 through U+2069 seem to be a good spot.

The rest of this document works under the assumption that for each of the new characters there is also a new Bidi_Class property value. This is because the current formulation of the UBA is in terms of bidi classes, and some existing implementations allow the caller to specify the class of each character in the input text instead of providing the actual characters. Nevertheless, adding new classes is problematic in light of the Unicode Stability Policy regarding the Bidi_Class property. Some alternative approaches to dealing with this are:

- Make character values (not just bidi classes) available to the algorithm, and restate the rules that affect the new characters in terms of characters and not Bidi_Class property values.
- State the algorithm in terms of a new property XBidi_Class (name TBD) which has the extra property values, and is otherwise identical to Bidi_Class.

Under such approaches, it would be possible to classify the new characters in the closest existing bidi class - LRI and FSI as LRE, RLI as RLE, and PDI as PDF. This might have the side-benefit of improving the rendering of documents containing the new characters by existing applications.

The **visual ordering within an isolate** is the same as if it were an embedding of the indicated direction, with the exception that it is *never* affected by the text outside the isolate. (This is not true for embeddings in some rare circumstances, such as when one embedding immediately follows another.)

More importantly, the **visual ordering outside the isolate** is the same as if the isolate were an ON-class character.

> Thus, the text “in Arabic, they are $_{FSI}$المغرب$_{PDI}$, $_{FSI}$الامارات العربية المتحدة$_{PDI}$, and $_{FSI}$العراق$_{PDI}$” would be displayed in an LTR context as:
>
> in Arabic, they are المغرب, الامارات العربية المتحدة, and العراق
>
> instead of the way it would be displayed if using RLEs and PDFs or no formatting:
>
> in Arabic, they are الامارات العربية المتحدة ,المغرب, and العراق
>
> (note the misordering of the names and the strangely placed commas).
>
> Similarly, “$_{LRI}$31$_{PDI}$/$_{LRI}$12$_{PDI}$/$_{LRI}$2012$_{PDI}$” would be displayed as “31/12/2012” in an LTR context and as “2012/12/31” in an RTL context. (With no formatting or with LREs and PDFs, this would be displayed as “31/12/2012” even in an RTL context.)
>
> Finally, “$_{RLI}$פיצה סגולה$_{PDI}$ (3 reviews)” would be displayed as
>
> פיצה סגולה (3 reviews)
>
> not as

3) פיצה סגולה reviews)
(which is how it would be displayed if using RLE and PDF or no formatting).

Isolates are skipped over when determining the base direction of a paragraph or a first-strong isolate. This is part of making an isolate behave as a neutral character for the purposes of the visual ordering of the content surrounding it.

> Thus, the paragraph "$_{LRI}$HTML$_{PDI}$ (ראשי תיבות של $_{LRI}$Hyper Text Markup Language$_{PDI}$) היא שפת תגיות ($_{LRI}$markup language$_{PDI}$) ליצירה ועיצוב דפי אינטרנט." (from the Hebrew Wikipedia article on HTML) would be interpreted as RTL even in the absence of a higher protocol for paragraph direction despite beginning with LTR text, and thus (correctly) displayed as:
>
> HTML (ראשי תיבות של Hyper Text Markup Language) היא שפת תגיות (markup language) ליצירה ועיצוב דפי אינטרנט.

Nesting is allowed. Embeddings and overrides are allowed to nest within isolates (and embeddings and overrides), and isolates are allowed to nest within isolates, embeddings and overrides. Just like embeddings and overrides, isolates count toward the existing nesting limit beyond which explicit formatting characters are considered invalid. The new PDI codepoint is used to terminate isolates instead of the existing PDF. As suggested by Martin J. Dürst, this makes sure that older applications that implement the UBA without isolate support (and thus ignore isolate formatting characters) do not misinterpret the end of an isolate as the end of an embedding or override in which the isolate is nested.

When the embeddings or overrides and isolates in a paragraph are **not properly nested**, we define the isolates to be "stronger". That is, the algorithm will ignore a PDF when a PDI is expected, and will have a PDI close all embeddings/overrides opened between the PDI and the FSI/LRI/RLI it closes.

> Thus, in "$_{LRE}$ ⋯ $_{RLE}$ ⋯ $_{FSI}$ ⋯ $_{PDF}$?! $_{PDI}$", the PDF is ignored, so the isolate continues to the PDI and includes the "?!".
>
> And in "$_{LRE}$ ⋯ $_{FSI}$ ⋯ $_{RLE}$ ⋯ $_{PDI}$?! $_{PDF}$", the PDI ends the scope of the RLE as well as the isolate, so the "?!" is outside the isolate. (The PDF winds up matching the LRE.)

When embeddings and isolates are properly nested, isolates being "stronger" makes no difference.

> Thus, in "$_{RLI}$ ⋯ $_{LRE}$ ⋯ $_{PDF}$ ⋯ $_{PDI}$" and "$_{RLE}$ ⋯ $_{LRI}$ ⋯ $_{PDI}$ ⋯ $_{PDF}$", the PDF closes the embedding, and the PDI closes the isolate. Neither is ignored.

We make isolates "stronger" than embeddings and overrides so that they can be used to isolate

the surrounding text from the effects of missing and extra PDFs inside the isolate.

> Thus, in "$_{\text{RLI·RLO·PDI}}$ the end" the isolate does not let the effects of the RLO leak out to make "the end" display as "dne eht".

We feel that this is an important enough advantage to outweigh the loss of the higher level of forward compatibility that would have been achieved by making isolates "weaker" (and thus result in new applications ignoring isolates in improperly nested paragraphs just like old applications). Thus, the proposal does not let older applications misinterpret nested isolates as ending embeddings and overrides - but only when the isolates are properly nested.

That isolates "stronger" than embeddings and overrides isolate the surrounding text from the effects of missing and extra PDFs inside them matches the behavior of CSS isolates in this respect (see following sections), which is another important consideration in their favor.

Another approach that has been suggested for treating improperly nested embeddings/overrides and isolates is a symmetric one: a PDF encountered when a PDI is expected would close the matching LRE/RLE/LRO/RLO as well as all isolates started in between, and a PDI encountered when a PDF is expected would close the matching LRI/RLI/FSI as well as all embeddings and overrides started in between. However, as pointed out by Asmus Freytag, this does not achieve the isolation of both missing *and extra* PDFs that "stronger" isolates provide.

> For example, with "stronger" isolates, "$_{\text{LRE}}$Drink $_{\text{RLI·PDF}}$א$_{\text{PDI}}$!$_{\text{PDF}}$" in an RTL context is displayed as "Drink א!" (because the extra PDF inside the isolate is ignored). On the other hand, with the symmetric approach, the extra PDF inside the isolate would terminate both the isolate and the LRE, resulting in the string displaying as "!א Drink".

Thus, the symmetric approach is also rejected in favor of "stronger" isolates.

As with embeddings and overrides, all **isolates are terminated at the end of a paragraph**.

Isolates in HTML and CSS

The CSS3 working draft specification (Writing Modes Level 3) has been expanded to provide isolation (and first-strong automatic detection for isolates) via new values of the unicode-bidi property: **isolate**, **plaintext**, and **isolate-override**. Each defines a different flavor of isolate (see more below).

The HTML5 specification employs these CSS features by default in various situations, e.g. unicode-bidi:isolate for the new <bdi> element and unicode-bidi:plaintext for <textarea dir="auto">.

The CSS/HTML definition for isolates is similar - *but not identical* - to the isolates proposed for Unicode above. Since it was formulated before Unicode isolates were envisioned (let alone available), it is based on a higher level protocol wherein:

- The content inside an isolate element is displayed as a separate paragraph or paragraphs (in the Unicode sense).
- Nevertheless, the isolate as a whole affects the content around it as if it were just a neutral character (U+FFFC).

While this higher-level protocol was already used to handle some pre-existing CSS constructs (such as elements with display:inline-block), isolates allowed its application to inline elements whose content may be wrapped just like the text around it. Applying this higher level protocol in this context has proven to be quite difficult to get right. And thus, while at least two of the major browser implementations, Mozilla and WebKit, now include prefixed implementations of CSS isolates, both have some significant bugs. The bugs have proven to be very difficult to fix - and the bugs in one implementation are not like the bugs in the other.

Proposed Changes to CSS Isolates

A major part of the motivation for the current Unicode proposal is to allow CSS implementations to use the proposed Unicode features instead of working around the Unicode algorithm. This should greatly simplify the implementation and improve interoperability.

For reference, let us look at how unicode-bidi:embed (a pre-existing, non-isolate value) uses existing Unicode embeddings: it has no effect on an element that is a containing block, and otherwise entails simulating an LRE or RLE at the start of the element (depending on the element's direction style) and a PDF at the end of the element.

We also note that the CSS specification already says that at a paragraph break inside an element that is not a containing block, the formatting codes appropriate for the end of the element are emitted before the paragraph break, and formatting codes appropriate for the start of the element are emitted again after the paragraph break. Thus, for

text1text2
text3text4

CSS is supposed to emit something like

$_{RLE}$text1$_{LRE}$text2$_{PDF \cdot PDF \cdot LF \cdot RLE \cdot LRE}$text3$_{PDF}$text4$_{PDF}$

Now, let us look at how the new isolate values would be implemented using the proposed Unicode features:

- unicode-bidi:isolate: No effect on a containing block. Otherwise, entails simulating an LRI or RLI at the start of the element (depending on the element's direction style) and a PDI at the end of the element.
- unicode-bidi:plaintext: On a containing block, prevents setting base paragraph direction to the element's direction style, leaving it up to P2 and P3. Otherwise, entails simulating an FSI at the start of the element and a PDI at the end.

- unicode-bidi:isolate-override: If the element is not a containing block, entails simulating an LRI or RLI at the start of the element (whichever would give the smaller level jump) and a PDI at the end of the element. Furthermore, whether the element is a containing block or not, entails simulating an LRO or RLO (again depending on the element's direction style) at the start of the element, after the LRI or RLI, and a PDF at the end of the element, before the PDI.

Please note that we have not proposed formatting characters that combine isolation with directional override for Unicode (the direct equivalent of CSS' unicode-bidi:isolate-override) because they are not likely to be very useful, and can be simulated by combining formatting characters as above, admittedly at the cost of using higher level number jumps.

At a paragraph break inside an isolate that is not a containing block, the CSS specification to emit the formatting codes appropriate for the end of the element before the paragraph break, and to re-emit the formatting codes appropriate for the start of the element after the paragraph break would still apply. Thus a
 within would first simulate a PDI, then the newline, and then an LRI. Similarly, a
 within would put an FSI after the newline. If the same occured for instead of a , i.e. in a containing block, no formatting codes would be involved, not even for unicode-bidi:plaintext (although of course the usual rules P2 and P3 would be apply to its paragraphs).

Unfortunately, however, such an implementation *would not agree with the current CSS specification*. Its behavior would differ from that implied by the current specification in two respects:

- A paragraph break within an isolate that is not a containing block would constitute a paragraph break in the content surrounding the isolate. The current specification implies that, although a paragraph break inside an isolate should start a new paragraph within the isolate, it should not affect the paragraph within which the isolate appears. The outside paragraph should continue to flow on the other side of the isolate, as if the isolate were a neutral character.
- Each isolate would start one or two levels higher than its surroundings. The current specification implies that it should start at level 0 or 1.

We are therefore working on changing the CSS specification for isolates to base it on the proposed Unicode isolate formatting characters as described above. This has the side benefit of simplifying the CSS specification.

Obviously, such a change in the CSS specification can not be made before Unicode isolates have been accepted into the Unicode standard, so our CSS specification modification proposal is currently hypothetical.

One could ask, however, why we are not proposing that Unicode isolates behave more like

current CSS isolate specification. The answer is that while it makes sense to have markup that can enclose multiple Unicode paragraphs, it does not make sense to have Unicode formatting characters whose effects span paragraph breaks (in a radical departure from existing Unicode bidi algorithm rules). Neither can it be said that the differences between the two flavors are unequivocally in favor of the CSS variety in terms of usability: 60 levels of embedding remain a purely theoretically limit that normal usage does not come close to approaching, and extending a paragraph across an inline isolate that in itself forms several paragraphs is not something that the reader of a document is likely to grasp.

Isolates vs Embeddings and Marks

Unicode has gotten along without isolation for quite some time. Everything that needs to be done can be done using the existing embedding characters (LRE and RLE and PDF) and directional marks (LRM and RLM). So, do we really needs isolates?

To illustrate the advantages of using isolates instead of embeddings and marks, let us use an extended example. To make it a little easier to understand, we will use uppercase English words instead of real RTL text for the duration of the example. Lowercase is reserved for what is intended to be LTR text.

The setting for our example starts with a database holding information about books, including their titles.

Let's say that there is a book in an RTL language whose RTL title (which could be approximately translated into English as “advanced c++ made simple”) is meant to be displayed like this:

ELPMIS EDAM c++ DECNAVDA

Using existing Unicode characters, it could be written (in logical order) as:

ADVANCED c++$_{\text{LRM}}$ MADE SIMPLE

or as:

ADVANCED $_{\text{LRE}}$c++$_{\text{PDF}}$ MADE SIMPLE

Both will be displayed as intended in an RTL context. The formatting characters are necessary to prevent the “c++” from coming out as “++c”.

Unfortunately, however, in the real world the database is going to be used by a simple-minded application whose authors have never heard of bidi and which simply plugs the values it gets out of the database into the text that it generates. If the application is LTR, the book title will be garbled (the same way for both versions above):

DECNAVDA c++ ELPMIS EDAM

This would not happen, however, if the book title used the proposed isolation characters:

ADVANCED $_{LRI}$c++$_{PDI}$ MADE SIMPLE

This would come out as intended in both LTR and RTL contexts.

The moral: *using isolation is more robust than using embeddings or marks*.

Of course, this will not always work perfectly. For example, a slightly shorter version of our title,

ADVANCED $_{LRI}$c++$_{PDI}$

will be displayed in an LTR context as

DECNAVDA c++

instead of the intended

c++ DECNAVDA

Seemingly, the isolates do no better here than the versions using embeddings and marks, i.e.

ADVANCED c++$_{LRM}$

and

ADVANCED $_{LRE}$c++$_{PDF}$

which are displayed in LTR with exactly the same problem.

Still, there is a difference. If we have another simple-minded application, this time an RTL one, that displays the book title followed by its date of publication, e.g. " (29 MAY 2008)", the traditional versions of our shortened book title do something terrible:

(2008 YAM c++ (29 DECNAVDA

The isolate version of the title, however, comes out as intended:

(2008 YAM 29) c++ DECNAVDA

Once again, this shows that isolates are more robust.

Of course, both of our simple-minded applications have a basic bidi bug: they plop book titles into their output without explicitly indicating where the book titles (and their directionality) begin and end, and leave it to the UBA to sort out the mess, which it can not always do. So, rather than look at buggy applications (where data that uses isolates behaves more robustly), we should really look at applications that try to do the right thing.

What is that right thing? Using existing embedding characters and directional marks, the application has to do the following:

- Determine the directionality of the book title. This is not always known or readily available, and may have to be estimated by looking at the text of the title. Quick, where do I find a library to do that for me?
- Wrap the book title in either LRE and PDF or RLE and PDF, depending on its directionality.
- Determine the directionality of the context in which the book title is to be displayed. Unfortunately, this also is not always readily available in the software layer doing the inserting.
- Insert an LRM after the book title if the context is LTR, or an RLM if the context is RTL. This "resets" the direction back to the context's, preventing the problems like the one with the date above, and has therefore been called "directional reset". It is also a technique of which few software developers are aware.

Taken together, this logic is *daunting*. Its complexity (and requirement to know the directional context) usually prevent all but the most advanced applications from displaying potentially opposite-diectionality text data correctly.

Now, compare the above with what the application has to do if using the new isolates:

- If the directionality of the book title happens to be known, wrap the book title in either LRI and PDI or RLI and PDI, depending on its directionality.
- Otherwise, wrap the book title in FSI and PDI.

If the application wants to be simple-minded, the first step can be (and usually is) skipped.

Thus, our simple-minded applications can always wrap a book title (e.g. "ADVANCED $_{LRI}$c++$_{PDI}$") in FSI and PDI (getting "$_{FSI}$ADVANCED $_{LRI}$c++$_{PDI·PDI}$"), and get the desired display in both LTR and RTL contexts, whether the title is followed by a date or not.

The moral: *using isolation is much easier than using embeddings and marks*.

The hope is that isolation will greatly lower the barrier for displaying opposite-diectionality data correctly.

Can’t LRE and RLE just become isolates?

Unfortunately, LRE, RLE and their HTML counterpart, the dir attribute (which CSS translates into LRE, RLE and PDF on inline elements) are not always used in the most logical manner, and changing their semantics to that of isolates would break many existing documents.

For example, one sometimes sees something like “ - “ put in between two short fields in an LTR page, so that when both of the fields around it contain RTL text, the first will still appears to the left of the other. Note that without the around the dash, and in the absence of isolation semantics, the two RTL fields would flow together (RTL overall) even if each one has a separate around it. And, more likely, neither one does. The reason someone would code a page in this manner, with no around the fields that need it, but with a around an innocent dash is twofold:

- They lack the technology or the inclination to determine the direction of the two fields, which can vary.
- They do not realize that an LRM between the two fields is a better solution to their woes (in the absence of isolation) than the around the dash.

Would applying isolation to this dash span break the page? It depends. If each of the two fields is itself wrapped in a , and isolation applies to those spans too, then everything will be fine (although, of course, the is no longer necessary around the dash when both of the fields have been isolated). On the other hand, if neither field is inside an element with dir=”rtl”, then isolating the dash span will break the page - the two fields will flow together.

Let us take another example. Sometimes, the non-isolate semantics of embeddings give a more desirable outcome when the page is done in a quick and dirty way. For example, once again, let's say we have two fields - a title and a body, and they are displayed in a single paragraph, in-line, perhaps with the title bolded. If they are both LTR in an RTL page, and the body is multi-line, it is definitely better to display the title to the left of the body. And that is exactly what one gets with current embed behavior, even if the title and body are each sitting in a separate dir=ltr element, e.g. Bidi Rocks! President Obama announced today... . If one applies isolation to those spans, however, the title will appear to the right of the body; isolation will break the desired layout of the page.

Now, the right way to do this page is to explicitly make the title and body a single directional unit that, at least when they both have the same direction, flows in that direction. And to make it even simpler, we can make that unit follow the direction of the body (since it is multi-line). So, one would have Bidi Rocks! President Obama announced blah blah... . (We have left the now unnecessary dir=”ltr” on the title to show where it would go when the title and body actually had different directions.) This gives the intended display under

both embed and isolate semantics. But it is more common to see the original, simpler structure, where isolation would indeed pose a problem.

To summarize, giving LRE and RLE isolation semantics would disrupt backward compatibility to an unacceptable level.

Isolates and the Level Direction Mark

Recently, an invisible LEVEL DIRECTION MARK (LDM) has been proposed that would dynamically take on the resolved direction associated with the current embedding level, i.e. behave either as an LRM or an RLM depending on the explicitly declared directional context.

How do isolates relate to this proposal?

First and foremost, the LDM, like LRM and RLM, is not designed to declare the overall direction of a phrase, which is the province of embeddings - and now isolates. No matter how many directional marks (LRM, RLM, or LDM) one inserts in or around “ADVANCED java MADE SIMPLE”, it still won’t come out as intended in an LTR context. One needs an embedding or isolate to do that. The LDM simply isn’t meant to deal with the issues that isolates tackle.

Its intended use is the “directional reset” technique mentioned in the preceding section, since it avoids the pitfall of having to know the directional context into which data is being inserted in order to decide between using an LRM or an RLM. This indeed makes it possible to use the technique in contexts where others it would be very difficult or impossible.

When using isolates, however, there is no systematic need to use LRM or RLM. Where there is no need for LRM or RLM, there is no need for an LDM either.

For example, let’s consider one of the stated use case for the LDM, "an Arabic numeric date of the form dd/MM/yyyy in which the fields should flow left-to-right (e.g. ٠٩/١٦/٢٠١١) in a left-right context (i.e. the date and perhaps some other Arabic text are in a mainly Latin-script paragraph), but should flow right-to-left (e.g ٢٠١١/١٦/٠٩) in a right-left context (e.g. a primarily Arabic-script paragraph)". This is achieved with isolates by putting each of the numbers (day, month, year) in an a separate isolate, e.g. $_{LRI}$٠٩$_{PDF}$/$_{LRI}$١٦$_{PDF}$/$_{LRI}$٢٠١١$_{PDF}$.

Furthermore, this way of handling the issue is more robust. What if the date above were to appear inside an Arabic sentence that was inserted into an English document *without being marked as RTL overall*? Lack of direction declaration is a very common practice that the UBA encourages by making text usually come out as intended even in a context of the opposite direction. Well, if an Arabic sentence used a date formatted with LDMs appeared in an LTR context, *the Arabic sentence would get rotated around the LDMs*. For example, what came out correctly in RTL as

وعلينا أن نفعل هذا من عام ٢٠١١/١٦/٠٩ حتى نهاية العقد الحالي.

would come out totally garbled in LTR as

٢٠١١. حتى نهاية العقد الحالي/١٦/وعلينا أن نفعل هذا من عام ٠٩

If, however, the same date were to be done using isolates, the Arabic sentence would come out exactly as intended (including the ordering of the date, which would be RTL to match the Arabic text around it), even in an LTR context.

To summarize, we believe that isolates handle the cases that LDM was meant to solve, and more. However, there is no actual conflict between the two proposals, and in the absence of isolates the LDM would definitely be useful.

Isolates and the Bidirectional Parentheses Algorithm

There is no conflict between isolates and the proposed modification to the UBA known as the Bidirectional Parentheses Algorithm (BPA). Their aims are orthogonal: the BPA aims to improve, sometimes, the display of text whose direction has not been explicitly declared; isolates aim to give a better way to explicitly declare the direction of text. Neither do the specific changes to the UBA contained in the two proposals seem to interact.

Detailed Changes to UAX #9

The following is a list of the specific changes that need to be made to UAX #9 (i.e. the UBA specification) in order to add support for isolate formatting codes.

The instructions for modifying UAX #9 appear below in yellow highlighting. The actual text to be inserted appears without highlighting.

Most of the changes are to section 3 (Basic Display Algorithm), and in particular to section 3.3.2 (Explicit Levels and Directions).

Section 2 (Directional Formatting Codes): Add the isolate codes

In the introductory paragraph, replace “two types of explicit codes” with “three types of explicit codes”.

The last two sentences of the introductory paragraph are currently a non-sequitor and interrupt between the subject matter preceding and following them. It would make sense to take them out and put them elsewhere if absolutely necessary.

In the second paragraph (“These controls all have the property Bidi_Control …”), replace “two groups” with “three ranges”.

Add a third range:
Explicit Bidi Controls for Isolates

U+????..U+???? LEFT-TO-RIGHT ISOLATE..POP DIRECTIONAL ISOLATE

One possible place to locate these characters would be U+2066 through U+2069.

In the paragraph beginning with “On web pages”, replace “with the values dir="ltr" or dir="rtl"” with “and the elements BDI and BDO”.

After “Characters within an embedding can affect the ordering of characters outside, and vice versa”, add the following sentence:
That is the precise difference between those codes and the explicit *isolate* codes. Characters within an isolate can not affect the ordering of characters outside it, or vice versa.

Add one more paragraph at the end of section 2.1 (“Explicit Directional Embedding”):
In most cases, using one of the explicit directional isolate codes will give the same or better results than an explicit directional embedding code.

Rename section 2.3 (“Terminating Explicit Directional Code”) to “Terminating Explicit Directional Embeddings and Overrides”.

In the introductory paragraph of 2.3, replace “last explicit code (either embedding or override)” with “last explicit embedding or override code whose effects have not yet been terminated”.

Add two new sections between 2.3 and 2.4.In the present document they are called 2.3+1 and 2.3+2, but when applied to UAX #9, the subsections should be renumbered:

2.3+1 Explicit Directional Isolates

The following codes signal that a piece of text is to be treated as directionally isolated from its surroundings. They are very similar to the embedding codes, and like them allow for nesting. (In fact, any explicit directional code can be nested within the scope of any other.) The difference between isolates and embeddings is that while an embedding roughly has the effect of a strong character on the ordering of the surrounding text, an isolate has the effect of a neutral like OBJECT REPLACEMENT CHARACTER (U+FFFC). Furthermore, the text inside the isolate has no effect on the ordering of the text outside it, or vice-versa.

In addition to allowing embedding text whose direction is the opposite of its surroundings without unduly affecting its surroundings, one of the isolate codes also offers an extra feature:

embedding text while guessing its direction from its constituent characters.

Abbr.	...	Name	Description
LRI		LEFT-TO-RIGHT ISOLATE	Treat the following text as isolated and left-to-right.
RLI		RIGHT-TO-LEFT ISOLATE	Treat the following text as isolated and right-to-left.
FSI		FIRST-STRONG ISOLATE	Treat the following text as isolated and in the direction of its first character with a strong implicit direction that is not inside a nested isolate.

The precise meaning of these codes will be made clear in the discussion of the algorithm.

2.3+2 Terminating Explicit Directional Isolates

The following code terminates the effects of the last explicit directional isolate code whose effects have not yet been terminated and restores the bidirectional state to what it was before that code was encountered.

It also terminates the effects of any explicit directional embedding or override codes that came after the last directional isolate code and whose effects have not yet been terminated.

Abbr.	...	Name	Description
PDI		POP DIRECTIONAL ISOLATE	Restore the bidirectional state to what it was before the last LRI, RLI, or FSI.

The precise meaning of this code will be made clear in the discussion of the algorithm.

The FSI, LRI, RLI, and PDI abbreviations should be added to http://www.unicode.org/Public/UNIDATA/NameAliases.txt.

Section 3 (Basic Display Algorithm): Hold off on specifying the order in which rules are applied.

In the "Resolution of the embedding levels" bullet, remove the sentence "Each rule is applied to

each of the values in sequence before continuing to the next rule". This is because the details of the order in which the rules are applied are given in later sections, and the formulation in the sentence above is potentially problematic for isolates.

Section 3.1 (Definitions): Add isolate-related definitions

BD2. Replace “Embedding levels are explicitly set by both override format codes and by embedding format codes” with: “Embedding levels are explicitly set by embedding format codes, isolate format codes and override format codes”.

After the example following BD7 add:

BD8. The *matching PDI* for a given FSI, LRI, or RLI is the one determined by the following algorithm:

- Initialize a counter to zero.
- Scan the text following the FSI, LRI, or RLI to the end of the of the paragraph while incrementing the counter at every FSI, LRI, or RLI, and decrementing it at every PDI.
- Stop at the first PDI, if any, before which the counter is already zero.
- If such a PDI was found, it is the matching PDI for the FSI, LRI, or RLI. Otherwise, there is no matching PDI for it.

Note that LRE, RLE, LRO, RLO and PDF characters are ignored when finding the matching PDI.

Add:

BD9. An isolating run sequence is an ordered set of level runs where:

- For every level run except the last one in the sequence, the last character in the level run is an FSI, LRI or RLI.
- For every level run except the first one in the sequence, the first character in the level run is the matching PDI of the FSI, LRI, or RLI at the end of the preceding level run in the sequence.
- If the first character of the first level run in the sequence is a PDI, it is not the matching PDI for any FSI, LRI, or RLI.
- If the last character of the last level run in the sequence is an FSI, LRI or RLI, it has no matching PDI.

As we will see, all the level runs in an isolating run sequence have the same embedding level.

Let’s take an example, writing the format codes in subscript to make it easier to read. Assuming that no *texti* contains format codes or paragraph separators, the paragraph “$text1_{FSI}text2_{LRI}text3_{PDI}text4_{PDI \cdot RLI}text5_{PDI}text6$” will contain the following isolating run sequences:

- “$text1_{FSI}$”, “$_{PDI \cdot RLI}$”, “$_{PDI}text6$”

- “*text2*$_{\text{LRI}}$”, “$_{\text{PDI}}$*text4*”
- “*text3*”
- “*text5*”

Add:
BD10. The *directional isolate status* is a Boolean value reflecting whether the current embedding level was started by an FSI, LRI, or RLI (as opposed to an LRE, RLE, LRO, or RLO).

This is followed by Table 3.

Replace the first row of Table 3 (defining N) with the following:

N	Neutral, Separator or Isolate formatting code (B, S, WS, ON, FSI, LRI, RLI, PDI)

Replace the last two rows (currently defining sor and eor, which we no longer use) with the following:

sos	The text ordering type (L or R) assigned to the virtual position preceding the first level run in an isolating run sequence.
eos	The text ordering type (L or R) assigned to the virtual position following the last level run in an isolating run sequence.

Section 3.2 (Bidirectional Character Types): Add the new types

The current grouping of LRE, RLE, LRO, and RLO as "Strong" and PDF as "Weak" doesn’t make any difference to the present algorithm or even make much sense. Instead, add a fourth group, "Formatting", consisting of LRE, RLE, LRO, RLO, PDF, LRI, RLI, FSI, and PDI.

Section 3.3.1 (The Paragraph Level), rule P2: Skip over isolates

After "In each paragraph, find the first character of type L, AL, or R", insert:
while skipping over any characters between an FSI, LRI or RLI and its matching PDI or, if it has no matching PDI, the end of the paragraph.

Section 3.3.2 (Explicit Levels and Directions): Include isolates

Modify the first paragraph to read as follows:

All explicit embedding levels are determined from the embedding, override, and isolate codes, by applying the explicit level rules X1 through X10. These rules are applied as part of the same logical pass over the input. The following variables are used during this pass:

- The current embedding level.
- The directional override status.
- The directional isolate status.
- A directional status stack where each entry consists of an embedding level, a directional override status, and an directional isolate status. The stack height is never over 60.
- A counter called the valid isolate count.
- A counter called the invalid isolate count.
- A counter called the invalid embedding count.

As each character is processed, these variables are adjusted and the character's explicit embedding level is set as defined by rules X1 through X9, given the variables' current values and the bidirectional type of the character.

Modify rule X1 by adding the following after "Set the directional override status to neutral.":
Set the directional isolate status to false. Set the stack to empty. Set the valid isolate count to zero. Set the invalid isolate count to zero. Set the invalid embedding count to zero.

Modify rules X2 through X5 as follows:

- Replace "If this new level would be valid" with "If this new level would be valid and the invalid isolate count and invalid embedding count are both zero".
- Replace "Remember (push) the current embedding level and override status" with "Push the current embedding level, override status, and isolate status on the directional status stack."
- Replace "Reset the current level to this new level" with "Reset the current level to the new level, reset the isolate status to false".
- Replace "If the new level would not be valid, then this code is invalid. Do not change the current level or override status" with "Otherwise, this RLE/LRE/RLO/LRO is invalid. Do not change the current level, override status, or isolate status. If the invalid isolate count is zero, increment the invalid embedding count by one."
- Replace "...59, 60 → 61; above 60, no change (do not change levels with RLE/RLO if the new level would be invalid)" with "...; 60 → 61 when the invalid counts are both zero, but no change in the current embedding level otherwise. Above 60, no change."
- Replace "...58, 59 → 60; above 59, no change (do not change levels with LRE/LRO if the new level would be invalid)" with "...; 59 → 60 when the invalid counts are both zero, but no change in the current embedding level otherwise. Above 59, no change."

After rule X5, start a new subsection:
Isolates

The rules of this subsection will appear between X5 and X6. To avoid confusion in this document, these rules are being called X5+*i*. Their actual numbering in the new version of UAX #9 is TBD. (Renumbering existing rules is problematic because external documents refer to them by their current numbers.)

Add:

X5+1. With each RLI, first set the RLI's explicit embedding level to the current embedding level, and then compute the least greater **odd** embedding level.

> a. If this new level would be valid and both the invalid isolate count and the invalid embedding count are zero, then this isolate code is valid. Increment the valid isolate count by one. Push the current embedding level, override status, and isolate status on the directional status stack. Reset the current embedding level to the new level, reset the isolate status to true, and reset the override status to neutral.
>
> b. Otherwise, this RLI is invalid. Increment the invalid isolate count by one, and leave all other variables unchanged.

For example, level 0 → 1; levels 1, 2 → 3; levels 3, 4 → 5; …; 60 → 61 when the invalid counts are both zero, but no change otherwise. Above 60, no change.

Add:

X5+2. With each LRI, first set its explicit embedding level to the current embedding level, and then compute the least greater **even** embedding level.

> a. If this new level would be valid and both the invalid isolate count and the invalid embedding count are zero, then this isolate code is valid. Increment the valid isolate count by one. Push the current embedding level, override status, and isolate status on the directional status stack. Reset the current embedding level to the new level, reset the isolate status to true, and reset the override status to neutral.
>
> b. Otherwise, this LRI is invalid. Increment the invalid isolate count by one, and leave all other variables unchanged.

For example, level 0, 1 → 2; levels 2, 3 → 4; levels 4, 5 → 6; …; 59 → 60 when the invalid counts are both zero, but no change in the current embedding level otherwise. Above 59, no change.

Add:

X5+3. With each FSI, apply rules P2 and P3 to the sequence of characters between the FSI and and its matching PDI, and if there is no matching PDI, then the sequence of characters between the FSI and the end of the paragraph, as if this sequence of characters were a paragraph. If P2 and P3 decide on paragraph embedding level 1, treat the FSI as an RLI in rule X5+1. Otherwise,

treat it as an LRI in rule X5+2.

Note that the current embedding level is *not* reset to the paragraph embedding level determined by P2 and P3. It goes up by one or two levels, as it would for an LRI or RLI.

Before rule X6, start a new subsection:
Non-formatting characters

Modify rule X6 by replacing “BN, RLE, LRE, RLO, LRO, and PDF” with “BN, RLE, LRE, RLO, LRO, PDF, RLI, LRI, FSI, and PDI”.

After rule X6, start a new subsection with an introductory paragraph:
Terminating Isolates

There is a single code, PDI, to terminate the scope of the last unterminated FSI, LRI, or RLI. It also terminates the scopes of all unterminated LREs, RLEs, LROs, and RLOs between the last unterminated FSI, LRI, or RLI and the PDI terminating it.

The rules below will appear between X6 and X7. To avoid confusion in this document, they are being called X6+*i*, but in the new version of UAX #9, all the X rules will be renumbered.

Add:
X6+1: With each PDI, perform the following steps:

a. If the invalid isolate count is greater than zero, decrement it by one. (This PDI matches an invalid FSI, LRI, or RLI.)
b. Otherwise, if the valid isolate count is zero, do nothing. (This PDI does not match any FSI, LRI, or RLI, valid or invalid.)
c. Otherwise, this PDI matches a valid FSI, LRI, or RLI:
    c.1. Reset the invalid embedding count to zero. (This terminates the scope of the invalid LREs, RLEs, LROs and RLOs encountered between the PDI and the FSI, LRI, or RLI matched by the PDI.)
    c.2. While the directional isolate status is false, keep popping the last entry from the directional status stack into the current embedding level, directional override status and directional isolate status. (This terminates the scope of the valid LREs, RLEs, LROs, and RLOs encountered between the PDI and the FSI, LRI, or RLI matched by the PDI, and leaves the directional isolate status true, in the scope of the matched FSI, LRI, or RLI.)
    c.3. Pop the last entry from the directional status stack into the current embedding level, directional override status and directional isolate status. (This terminates the scope of the matched FSI, LRI, or RLI.)
    c.4. Decrement the valid isolate count by one.
d. In all cases, assign the current embedding level resulting from the steps above to the

PDI.

Please note that these rules guarantee that the explicit embedding level assigned to an FSI, LRI, or RLI is always the same as that assigned to the matching PDI.

Replace the introductory paragraph of the “Terminating Embeddings and Overrides” subsection:
There is a single code, PDF, to terminate the scope of the last unterminated LRE, RLE, LRO, or RLO. However, it only terminates it if the last unterminated FSI, LRI, or RLI came before the last unterminated LRE, RLE, LRO, or RLO. Otherwise, it is ignored.

Replace rule X7:
X7. With each PDF, perform the following steps:

a. If the invalid isolate count is greater than zero, do nothing. (This PDF either terminates the scope of an invalid LRE, RLE, LRO, or RLO within the scope of an invalid FSI, LRI or RLI, or does not match any LRE, RLE, LRO, or RLO.)
b. Otherwise, if the invalid embedding count is greater than zero, decrement it by one. (This PDF terminates the scope of an invalid LRE, RLE, LRO, or RLO that does not fall within the scope of an invalid FSI, LRI or RLI.)
c. Otherwise, if the directional isolate status is false, pop the last entry from the directional status stack into the current embedding level, directional override status and directional isolate status. (This PDF terminates the scope of a valid LRE, RLE, LRO, or RLO.)
d. Otherwise, do nothing. (This PDF does not match any LRE, RLE, LRO, or RLO.)

After rule X7, start a new subsection with an introductory paragraph:
End of Paragraph

The state is completely reset at the end of a paragraph.

Reword rule X8:
X8. With each paragraph separator, set its explicit embedding level to the paragraph embedding level of the preceding paragraph, then reset the state of the variables as described in rule X1, using the paragraph embedding level of the following paragraph.

Add a note to rule X9:
Note that FSI, LRI, RLI, and PDI codes are *not* removed.

After rule X9, start a new subsection with an introductory paragraph:
Resolving the Levels

The preceding rules have resulted in explicit embedding levels for the characters. These explicit embedding levels now need to be adjusted to the final resolved levels on the basis of the

characters' bidirectional type values. Before doing so, the characters are logically divided into a set of isolating run sequences as defined above (BD9). Since the last character in each level run except the last one in an isolating run sequence is an FSI, LRI, or RLI, and the first character of the following level run is the matching PDI, and since an FSI, LRI, or RLI always has the same explicit embedding level as its matching PDI, all the characters in an isolating run sequence have the same explicit embedding level.

Replace rule X10:

X10. The remaining rules are applied to the bidirectional type values of the characters in each of the isolating run sequences. The order in which isolating run sequences are processed relative to one another makes no difference. Each rule is applied to each of the values in an isolating run sequence successively before continuing to the next rule. When applying a rule to an isolating run sequence, the last value of each level run except the last one in the isolating run sequence is treated as if it were immediately followed by the first value in the next level run in the sequence.

For example, if a rule asks to search backward from an instance of a European number until the first strong type is found, the search skips over the characters in between successive level runs in the sequence.

For each isolating run sequence, we determine a *start-of-sequence* (sos) and *end-of-sequence* (eos) type, which is either L or R depending on the higher embedding level between that of the isolating run sequence and, for the sos, that of the character preceding the first character of isolating run sequence, and for the eos, that of the character following the last character of the isolating run sequence. At the start or end of the paragraph, we compare the embedding level of the isolating run sequence with that of the paragraph. In all cases, if the higher level is odd, the type is R; otherwise, it is L.

Let's take a look at some examples, each of which is assumed to be a paragraph with base level 0 where no *texti* contains format codes or paragraph separators. To make the examples easier to read, the format codes are written in subscript.

Example 1: "*text1*$_{\text{RLE}}$*text2*$_{\text{LRE}}$*text3*$_{\text{PDF}}$*text4*$_{\text{PDF·RLE}}$*text5*$_{\text{PDF}}$*text6*"

- "*text1*": level 0, sos L, eos R
- "*text2*": level 1, sos R, eos L
- "*text3*": level 2, sos L, eos L
- "*text4text5*": level 1, sos L, eos R
- "*text6*": level 0, sos R, eos L

Example 2: "*text1*$_{\text{RLI}}$*text2*$_{\text{LRI}}$*text3*$_{\text{PDI}}$*text4*$_{\text{PDI·RLI}}$*text5*$_{\text{PDI}}$*text6*"

- "*text1*$_{\text{RLI}}$", "$_{\text{PDI·RLI}}$", "$_{\text{PDI}}$*text6*": level 0, sos L, eos L
- "*text2*$_{\text{LRI}}$", "$_{\text{PDI}}$*text4*": level 1, sos R, eos R
- "*text3*": level 2, sos L, eos L

- “*text5*”: level 1, sos R, eos R

Example 3: “*text1*$_{RLE}$*text2*$_{LRI}$*text3*$_{RLE}$*text4*$_{PDI}$*text5*$_{PDF}$*text6*”

- “*text1*”: level 0, sos L, eos R
- “*text2*$_{LRI}$”, “$_{PDI}$*text5*”: level 1, sos R, eos R
- “*text3*”: level 2, sos L, eos R
- “*text4*”: level 3, sos R, eos R
- “*text6*”: level 0, sos R, eos L

Remove the current example after X10 and the paragraoh after it (“For two adjacent runs, the eor of the first run is the same as the sor of the second.”).

Section 3.3.3 (Resolving Weak Types): Use isolating run sequences

Replace the first paragraph “Weak types are now resolved … the type assigned to sor or eor is used”:

Weak types are now resolved one isolating run sequence at a time. At the start of an isolating run sequence, where the type of the character preceding the first character of the first level run is required, the type assigned to sos is used. At the end of an isolating run sequence, where the type of the character following the last character of the last level run is required, the type assigned to eos is used.

Throughout the section, replace each instance of:

- “sor” with “sos”
- “eor” with “eos”
- “level run” with “isolating run sequence”

Section 3.3.4 (Resolving Neutral Types): Use isolating run sequences

Replace the first paragraph “Neutral types are now resolved ... the type assigned to sor or eor is used.”:

Neutral types are now resolved one isolating run sequence at a time. At the start of an isolating run sequence, where the type of the character preceding the first character of the first level run is required, the type assigned to sos is used. At the end of an isolating run sequence, where the type of the character following the last character of the last level run is required, the type assigned to eos is used.

Throughout the section, replace each instance of:

- “sor” with “sos”
- “eor” with “eos”
- “level run” with “isolating run sequence”

Section 3.4 (Reordering Resolved Levels): Treat isolate format codes as whitespace

In Rule L1, items 3 and 4, replace “Any sequence of whitespace characters” with “Any sequence of whitespace characters and/or isolate format codes (FSI, LRI, RLI, and PDI)”.

In the paragraph introducing the examples, replace “with Bidi Controls and BN characters removed” with “with BN characters and some Bidi Controls removed”.

Section 4.2 (Explicit Formatting Codes): Add intermediate class

Replace “three classes” with “four classes”. Split the last class (“Full bidirectionality”) with two:

- Non-isolate bidirectionality. The implicit Bidirectional Algorithm, the implicit directional marks, and the explicit non-isolate directional formatting codes are supported: RLM, LRM, LRE, RLE, LRO, RLO, PDF.
- Full bidirectionality. The implicit Bidirectional Algorithm, the implicit directional marks, and all the explicit directional formatting codes are supported: RLM, LRM, LRE, RLE, LRO, RLO, PDF, FSI, LRI, RLI, PDI.

Section 4.3 (Higher-Level Protocols): Allow isolate emulation

Modify HL3 to include isolates:

- Replace the title (“Emulate directional overrides or embedding codes”) with “Emulate directional formatting codes”.
- In the body, replace “a directional override or embedding” with “a directional embedding, isolate, or override”.

Section 5.2 (Retaining Format Codes): Use isolating run sequences

Throughout the section, replace each instance of:

- “sor” with “sos”
- “eor” with “eos”
- “level run” with “isolating run sequence”

Note: I have no idea what “In rule X10, assign L or R to the last of a sequence of adjacent BNs according to the eor / sor, and set the level to the higher of the two levels” means. It probably needs editing.

Section 5.5 (Usage): Add isolate usage suggestions

Needs work. We may want to deprecate the embedding formatting codes in favor of isolates.

Section 5.6 (Separating Punctuation Marks): Also suggest isolates

Needs work.

Document History

July 24, 2012	Discussion of “symmetric” vs “stronger” isolates.
July 22, 2012	Isolates terminated by PDI, not PDF. Algorithm details (changes to UAX #9). Relationship to other proposals. Editorial rewrite.
May 7, 2012	Original draft