Title:	**Proposal to Encode SENTENCE JOINER and SENTENCE NON-JOINER**
Revision:	**version 2**
Source:	**Unicode Localization Interoperability Technical Committee**
Contact name:	**Arle Lommel (arle.lommel@gmail.com)**
Date:	**2012-05-09**

1. Summary

This document proposes the addition of two new characters to the **Supplemental Punctuation** block of the Unicode Basic Multilingual Plane (2E00–2E7F).

The SENTENCE JOINER character is used to indicate that text should *not* be segmented at an apparent Unicode Standard Annex #29 (UAX #29) sentence break point (e.g., after a full stop [.] that terminates an abbreviation rather than a sentence). It provides a way for plain text to indicate proper sentence-level segmentation in those cases where UAX #29 alone is insufficient.

> **Example**: *UAX #29 would segment the string "We saw Mr. Smith three nights ago" as two segments—"We saw Mr." and "Smith last night"—even though they are in fact one sentence. Inclusion of the SENTENCE JOINER after "Mr." and before " Smith" would instruct a UAX #29-compliant process to ignore the apparent break point.*

The SENTENCE NON-JOINER character is used to indicate the existence of a *sentence boundary* in instances where UAX #29 would normally not provide one. While instances were non-joining behavior is required are likely to be considerably rarer than those where joining behavior is required, they are needed in instances such as raw speech-to-text output, which lacks punctuation or in cases where normal capitalization rules do not apply.

> **Example**: *UAX #29 would normally not segment the following string properly because of the unusual capitalization in the second sentence: "Smart phone penetration rates are increasing dramatically. iPhones usage now makes up over half of all mobile web traffic." It could also be used to indicate sentence boundaries in contexts where the punctuation or capitalization cues relied upon by UAX #29 are unreliable (e.g., in raw output from speech-to-text systems).*

These characters are primarily intended for use in conjunction with UAX #29: while UAX #29 defines segmentation for Unicode text, by itself it is incomplete because it may generate false positives or (less frequently) miss sentence boundaries and does not provide a way to indicate user-generated sentence boundaries. These characters address this deficit for plain-text environments and also provide a way for linguistic processes that provide sentence-level segmentation, whether based on UAX #29 or not, to manually indicate their results in plain-text environments.

NB: The W3C's I18n Core Working Group has requested that markup solutions be recommended for use in XML documents and the characters proposed in this document be recommended for use in plain-text environments only, similar to the recommendation for other Unicode control characters. Therefore, this proposal recommends that markup-based formats implement appropriate markup for sentence-level segmentation and that these characters be used for plain text.

2. Background

This proposal emerges from efforts of the Unicode Localization Interoperability (ULI) Committee to provide standard resources to improve text segmentation on a per-language basis. Proper segmentation of text is important for a number of critical text-processing functions such as:

- Content management and retrieval. Systems that reuse content suffer from decreased efficiency when they are not able to serve up proper sentences due to segmentation errors.
- Content parsing for NLP purposes. Incorrect segmentation can lead to failure of processes that assume grammatical input.
- Controlled authoring. If text is incorrectly segmented it can decrease the effectiveness of controlled authoring systems.
- Text summarization. An aggregator of news streams might display the first sentence of articles to aid readers in determining what content is relevant, but incorrect breaks (e.g., after an abbreviation) result in users receiving ill-formed content.
- Translation. Modern computer-assisted translation systems work on "segments" (usually sentences) as a basic unit. Inconsistency in sentence-level segmentation results in decreased efficiency and failure to reuse previously translated assets.

Sentence boundaries are often determined using natural language processing (NLP) approach (e.g., a UAX #29-based parser or a syntactic parser that analyzes the grammar of a text to determine boundaries) or may be inserted manually or overridden by users (language- processing tools generally include options to allow users to split or merge segments into sentences when the tools propose incorrect boundaries). In most cases (and in agreement with UAX #29) sentence boundaries are found through the use of regular expressions, generally—at least for Western languages—something like the following (grossly simplified) example:

```
\w\p{P}*\.\p{P}*\s+\w\p{Lu}
```

(i.e., a word character optionally followed by punctuation characters followed by a period optionally followed by punctuation characters followed by white space followed by an upper-case letter character) (see UAX #29 for a more accurate and complete description). However, such segmentation rules will generate false positive boundaries for most Western languages[1] since boundary conditions may be ambiguous, as in the following example:

```
Mrs. Smith and Mr. Jones ate lunch at Mme. Flaubert's apartment.
```

In this case a simple UAX #29 processor would split the text inappropriately into four "sentences":

Mrs.	Smith and Mr.	Jones ate lunch at Mme.	Flaubert's apartment.

This example exemplifies the major obstacle to automatic segmentation of text in most Western languages: the full stop (.) character is semantically ambiguous: it may indicate the end of a sentence, a decimal marker or digit separator (depending on the language), to mark the end of an acronym or abbreviation, within initialisms (e.g., "U.S.S.R"), or as a separator in numerous technical usages (e.g., within IP addresses). While some of these uses are easily accounted for (e.g., the decimal usage), abbreviations are particularly problematic since they can lead to the spurious recognition of apparent sentence boundaries, as shown above.[2] While UAX #29 describes general principles for dealing with full stops, it does not address how to handle exceptions around abbreviations.

---

[1] Determining sentence-level segmentation in languages not based on Western scripts is generally easier than in Western languages because the punctuation used to end sentences is generally not ambiguous. For example 。 (U+3002) is used in Japanese unambiguously to end sentences and does not share the semantic ambiguity of the Western full stop (U+002E). (By contrast, determining word-level segmentation may be substantially more difficult in East Asian languages such as Thai, Chinese, and Japanese.)

[2] It should be noted that in some cases it may be desirable as well to segment on other characters not addressed in UAX #29: for instance, some translation memory engines segment on semicolons (;).

It is in fact impossible to determine the location of sentence boundaries based on a finite set of regular expressions without error since the same string may be ambiguous, e.g.,:

```
John brought the package to Mulholland Dr. Friday...
```

which will be segmented differently depending on the contexts in which it appears, e.g., consider the following (where # shows sentence boundaries):

1. John brought the package to Mulholland Dr.# Friday was the deadline for delivery.# (two sentences)
2. John brought the package to Mulholland Dr. Friday night when it was raining.# (one sentence)
3. John brought the package to Mulholland Dr.# Friday night when it was raining he was happy.# (two sentences)

While such ambiguous cases are likely to be relatively infrequent (and the two-sentence versions are awkward), they do demonstrate the need for a way to indicate sentence boundaries within text (or alternatively, to indicate that a potential segmentation point returned by UAX #29 or another method should not be used). In fact, one common function in text translation environments is the ability to merge or split segments of text when the processor makes a mistake.

2.1 Addressing Sentence Segmentation Faults

At present there are two common solutions to sentence-level segmentation faults, both of which may be used in combination with each other: (1) systems may provide users with ways to manually correct faults by joining or splitting sentence candidates; (2) in the translation and NLP environments it is common to use a set of *exceptions*, generally regular expressions for instances in which segmentation behavior should be suppressed. The latter approach is that taken by the Segmentation Rules eXchange (SRX) standard,[3] developed by the former Localization Industry Standards Association.

As ULI investigated the best ways to standardize segmentation data, the committee realized that there is currently no purely Unicode mechanism to indicate overrides of default UAX #29 behavior. While there are markup solutions, there is no standard XML-based solution and existing solutions are not suitable for plain text. The lack of override characters mean that when multiple tools are required to segment text (a common situation, especially in translation environments) there is no way to ensure that manual overrides or corrections are recorded, leading to inconsistency and increased processing burdens. This issue is especially crucial for standardized segmentation data being developed by ULI for inclusion in CLDR.

One of the primary areas where sentence-level segmentation has an impact is in *translation memory* reuse. Translation memory refers to a technology that stores aligned databases of texts (divided into *segments,* usually understood to be sentences) and their translations to support the reuse of previously translated texts. When different translation memory tools are used, or when users customize segmentation routines, it can lead to inconsistent segmentation that leads to a loss of reuse of

---

[3] Segmentation Rules eXchange is a standard developed by the now-defunct Localization Industry Standards Association (LISA). SRX defines a syntax for describing regular expression-based segmentation processes. SRX rulesets consist primarily of rules that describe *no-break rules* (exceptions to UAX #29 where segmentation should be inhibited, e.g., after certain abbreviations) along with *break rules* (which describe where to segment text). The latter allows the rules to describe rules beyond those supported by UAX #29, such as breaking at semicolons, as mentioned above.

translation memory data. In some cases losses caused by segmentation differences and by inconsistencies in how formatting codes are handled can lead to double-digit percentages of loss in efficiency. If tools, however, could use the same sentence- level segmentation at all stages, much of this loss could be eliminated.

As a result it is desirable for there to be a plain-text method for indicating the actual sentence segmentation (automatic, manual, or a combination of the two) that was derived from or applied to a given text so that other processes can operate on the same basis.

In English (and other European languages) abbreviated forms are particularly an issue for so- called *prefixing* abbreviations, those that are likely to occur before a name (e.g., "Dr." for *Doctor* or "Mr." for *Mister)*. Other abbreviations are less likely to cause problems (e.g., "etc.") but are often ambiguous when followed by names or other capitalized items and thus may require manual correction. In German, by contrast, there are likely to be more issues since nouns are normally given an initial capitalization, so general-purpose abbreviations are more likely to create issues. For example, the German abbreviation "z.B." (zum *Biespiel* 'for example') is regularly followed by nouns, as in "Alltägliches - z.B. Zitronenduft im Winter."[4] Thus the exact nature of the rules is likely to vary by language, even beyond the specific list of abbreviations.

The general approach to solving the problem of false positives is to use *exception lists,* as described above. However, these lists are by definition incomplete—language adds new abbreviations regularly—and are subject field- and language- specific. Even in cases where exception lists are complete, there will be ambiguous cases, such as "Dr." (*Doctor* or *Drive)—the* first is unlikely to mark a sentence boundary while the second, when followed by an uppercase generally will mark a sentence boundary—or "No." (a negative statement or a common abbreviation for *number)*. While improved regular expressions can address some cases, there is always a residuum which cannot be handled.

Since texts may be operated upon by a variety of tools (authoring, translation, publishing, content management, etc.), it is desirable to preserve information on how text has been segmented into sentences (or had its segmentation modified) so that the entire chain of tools can be aware of how text was segmented. While there are markup-based solutions (e.g., encapsulation in XLIFF or TMX format), these are not suitable for plain-text environments or any environment where markup is likely to be ignored or misinterpreted.

By contrast, a plain-text character-based solution allows for the (optional) incorporation of segmentation information into plain-text environments. This mechanism would allow for the information to be included or passed along in any plain-text environment without the need for a markup-based solution.

Research by the ULI committee has shown that the impact of segmentation differences can be quite high for organizations dealing with information from heterogeneous sources. In many cases only careful engineering solutions allow for interoperability, while a standard way to indicate sentence-level segmentation could reduce the burden significantly.

3. Alternatives Considered and Rejected

The ULI committee considered a number of alternatives, as described below. These initially focused on just the requirement for a SENTENCE NON-JOINER, although it was quickly realized that the joiner character is considerably more important in the context of UAX #29. As a result, this proposal addresses the need for a pair of characters.

---

[4] Taken from a blog: http://majorahn.blogspot.com/2012/01/alltagliches-zb-zitronenduft-im-winter.html.

3.1. Markup-Based Solutions

One solution considered was to mandate the use of markup for sentence boundaries. This solution was rejected because markup-based solutions lack portability: if a process does not support the particular markup solution, it will not be able to use it. If a tool or process does not support namespaces (either in general or a particular namespace) it will not be able to use a markup-based solution. Non-XML processes also would not be able to use a markup- based solution (e.g., a user of a non-XML-based authoring tool could not indicate segmentation exceptions using a markup-based solution). This solution was thus rejected as too limiting for general use, even though markup-based solutions may be preferable in many instances.

(As noted previously, feedback from the W3C indicates that the UTC should recommend the use of XML markup-based solutions within XML documents rather than character-based solutions.)

3.2. Extension of Existing Characters

The ULI committee considered a number of existing characters that might serve the purpose. Ultimately we rejected all of these for reasons noted below. However, in general, testing by Kevin Lenzo of Apple indicated that any existing characters with similar semantics created a strong chance of breaking existing processes. For example, Kevin tested WORD JOINER (U+2060), NO-BREAK SPACE (U+00A0), and INVISIBLE SEPARATOR (U+2063) by modifying a UAX #29 parser. All of these characters created major problems because of unintended side effects. In addition, even if UAX #29 were modified as appropriate, using existing control-type characters would require testing in all possible contexts for many tools, making the barrier to implement the new semantics very high and creating problems for older tools interacting with new data. This approach would thus undermine stability in UAX #29.

3.2.1. Suggested for Non-Joiner Usage

3.2.1.1. U+001F

This character has the advantage of occurring only rarely in plain-text environments and having a legacy use as a separator. After discussion, this option was rejected because, as a control character, it would be problematic for data entry and in plain-text environments where control characters are rejected. In addition, because content creators cannot always anticipate every possible use of their content or all details of tools that work with it, there is a strong possibility that text containing these characters would end up rendered in an XML format (e.g., put into a CDATA section for transport), where the use of a control character would render the file invalid. Even if the strong recommendation is to use a markup-based solution for XML, we believe it would be inadvisable to use a control-range character knowing that its inadvertent inclusion in an XML file would render the file invalid.

3.2.1.2 U+2063

Mark Davis suggested that U+2063 (INVISIBLE SEPARATOR) be used to indicate sentence boundaries. This suggestion is attractive because the semantics of U+2063 are largely compatible with the notion of a sentence boundary marker and the character is out of the Unicode range that is problematic should the character be included in text in an XML-based format (as noted in the previous section). Like U+001F, however, it runs into the difficulty of what to do with native instances of the U+2063 that occur in text. While these are likely to be quite rare, they would lead to confusion when they do occur. In addition, testing by Apple indicated that this character created undesirable side effects in testing.

3.2.1. Use of ZWJ and ZWNJ

ULI also considered extending the use of ZERO-WIDTH JOINER (ZWJ, U+200D) and ZERO-WIDTH NON-JOINER (ZWNJ, U+200C) to serve as sentence joiners and non-joiners respectively. Superficially these are a good match since they have appropriate semantics and the existing use (to control typographic features of ligature formation) is in complementary distribution with the intended use as sentence joiners and non-joiners (i.e., they are unlikely to occur in situations where the intended function would

be ambiguous). However, testing at Apple[5] showed that using these characters would require substantial modification of UAX #29 in ways that might have unpredictable results.

3.3. Use of Proprietary Strings

One alternative is to use a proprietary string to indicate a sentence boundary or a point where a boundary should be suppressed. For example the string "#§#" or a similar sequence unlikely to appear in actual texts could be inserted into the text as appropriate. This is the approach taken most often at present in NLP applications where XML markup is not used (e.g., "#" is sometimes used to indicate various types of segment boundaries).

Such usage has the advantage of being achievable using the current Unicode repertoire with no extension of character semantics. It is problematic, however, in that the sequence will be interpreted literally by processes unaware of its existence and that it must be stripped from the text prior to publication, thus eliminating the possibility of reuse in any subsequent processes that may arise (for example, search results would be unable to reference sentence boundaries when examining published texts). In addition, many tool vendors would have to agree on a single string-based solution for the advantages to be universally available, an unlikely scenario.

3.4. Conclusion

Based on ULI analysis and experimentation by Kevin Lenzo, we were unable to locate any characters with existing semantics suitable for the required purpose where overloading their semantics would not create undesirable behaviors that undermine the stability of Unicode, particularly UAX #29. It is our contention that the addition of a pair of new characters represents the least disruptive solution to meet the needs of CLDR and the various usages scenarios described.

4. New Characters and their Usage

This proposal is for the inclusion of two new characters in the Supplemental Punctuation block of the Basic Multilingual Plane. Although these characters logically belong with characters like ZWNJ (U+200C) and ZWJ (U+200D) in the General Punctuation block in terms of their function, there is more room in the Supplemental Punctuation block. The characters are defined as follows

---

[5] Kevin Lenzo described the testing procedure as follows:

The process was:

(1) Assume this character is used for the sentence non-break character.
(2) Insert it to the right of any punctuation, which would otherwise trigger a UAX 29-based sentence break.
(3) Run the result through the current ICU sentence break iterator.
(4) Check if we got what we wanted.

In each of these either did not stop breaks from being inserted, due to existing expressions involving FORMAT (and EXTEND) characters, which move the break to the right of these characters, rather than suppress it.

As an alternative, an arbitrary character is chosen to be a placeholder for the no-break, and the sentence break rules are modified to pass through any character to the left of the no-break character. When this is done, and the character is used for the sentence non-break character in the process above, and the result is as expected for the limited tests to-date.

1. SENTENCE JOINER: used to indicate that text should not be segmented at an apparent segmentation point or to indicate that two strings have been joined manually into one sentence (e.g., in a translation process).
2. SENTENCE NON-JOINER: used to indicate the existence of a sentence boundary.

While these characters would normally be invisible in plain text, the following proposed glyphs could be used in situations where visualization is desirable:

Character	Abstract glyph	Text-based glyph
SENTENCE NON-JOINER	] | [	SNJ
SENTENCE JOINER	] _ [	SJ

4.1. Usage Examples

The following are examples of tools or processes that might use these characters:

- The case that inspired this proposal is that of *translation memory* technology. This technology divides text into segments that are compared against a database of previously translated sentences to automate reuse of existing translations. Both the SENTENCE JOINER and SENTENCE NON-JOINER would be used to allow texts to indicate where segmentation corrections have been implemented and ensure that they are retained in subsequent processes.
- A web tool might extract the first sentence of an article to display in search results: if the tool selects something other than the first sentence (e.g., a portion of a sentence), it will result in an inconsistent user experience. In this case the SENTENCE JOINER would enable the tool to grab the first sentence in its entirety.
- NLP researchers would benefit from having a standard way to indicate segment boundaries using the SENTENCE NON-JOINER in place of ad hoc methods.

The SENTENCE JOINER would also serve the purpose in translation memory or other NLP environments that use aligned texts of showing instances where one sentence in one language corresponds to more than one in another language (e.g., one language says "She saw the man who had carried the package" while another renders it more like "She saw the man. He had carried the package.").

To show how both characters would work in tandem, the following examples, presented earlier, are shown with [SJ] and [SNJ] to indicate the location of the proposed characters. (Note that in the cases of these examples, it is highly likely that human intervention would be required to disambiguate them and arrive at the appropriate sentence boundaries.)

1. John brought the package to Mulholland Dr. [SNJ]Friday was the deadline for delivery. (two segments)
2. John brought the package to Mulholland Dr. [SJ]Friday night when it was raining. (one segment)
3. John brought the package to Mulholland Dr. [SNJ]Friday night when it was raining he was happy. (two segments)

Note that Examples 1 and 3 would both be segmented properly by UAX #29, so in many cases it would be safe to omit the [SNJ] character. However, including it might be desirable as many processes that use Segmentation Rules eXchange (SRX) to override the default UAX #29 behavior would treat "Dr. Friday" as a name and thus would not separate the segments.

In Example 2, UAX #29 would incorrectly split the segment into two pieces, but having [SJ] in place would override that behavior and tell the UAX #29-based processor not to split the segment, thus maintaining the proper linguistic unit (a sentence).

In general we foresee two general ways these characters might be used:

1. **As a supplement to UAX #29.** The characters could be used to indicate overrides to UAX #29 results. [SJ] and [SNJ] would be used in instances where deviation from UAX #29 is needed, but would otherwise leave the text alone. In this scenario, the characters can be seen as adjuncts to UAX #29 that allow for manual control where needed.
2. **As stand-alone segmentation markers.** In this scenario a SENTENCE NON-JOINER would be inserted at every point in the text where a sentence boundary is inserted by a process. In this scenario no subsequent UAX #29 processing is required because the process that inserts the characters is making a *positive declaration* about the location of all segment boundaries. This usage would be particularly useful for interaction with any tools that do not implement UAX #29.

Which scenario is used in any given instance is beyond the scope of this proposal and full process interoperability around these characters would require the exchange of information about texts to be processed.

4.1.1. Note on Placement
Placement of the proposed characters would require some sensitivity. The joiner would generally be placed immediately after the punctuation mark that triggers the problematic break behavior. The non-joiner could be placed anywhere, but should generally be placed to the right of any white space that follows the sentence it is intended to delimit (i.e., trailing white space belongs with the previous sentence, not the following sentence).

4.2. Impact on Search
Use of these characters creates potential problems for search algorithms since "downstream" users (i.e., people who use text after the proposed characters have been inserted but who are not themselves responsible for them) may not know the characters are in use (especially since the default would be to treat them as invisibles). As a result users may find unexpected results (much as they might when searching in text containing other invisible characters like U+200B [ZWS]). Note that both characters should be found by the \s (the regular expression whitespace metacharacter), but inclusion in search algorithms will take some time.

5. Impact on UAX #29
If the two proposed characters are accepted, section 5.1 (particularly Table 4) will need to be modified to reflect these characters.

The text should be modified as follows:

> *Do not break after ambiguous terminators like period, if they are immediately followed by a number or lowercase letter, if they are between uppercase letters, if the first following letter (optionally after certain punctuation) is lowercase, if the first following character is a SENTENCE JOINER (possibly following a space), or if they are followed by "continuation" punctuation such as comma, colon, or*

semicolon. For example, a period may be an abbreviation or numeric period, and thus may not mark the end of a sentence.

Modify rule SB4, as follows (so that SENTENCE NON-JOINER indicates a break under all conditions):

Sep | CR | LF SNJ ÷

Add a rule SB8b, as follows (to allow SEN-J to override normal break conditions):

(STerm | ATerm) Sp* × SEN-J × Sp*

In addition, the Sentence Break Property file (http://www.unicode.org/Public/UNIDATA/auxiliary/ SentenceBreakProperty.txt) will need to be modified to match the prose changes made above. (ULI will require appropriate expert assistance in modifying this file.)

Section 6 of UAX #29 will also require modification to account for these characters, and ULI will need to consult with a UAX #29 expert to make the appropriate changes.

NB: Because these character would typically be invisible and carry no formatting information, it is not anticipated that they will impact any layout or formatting aspects of the Unicode standard or of plain text in general.

6. Unicode Character Properties

- ````????;SENTENCE NON-JOINER;Cf;0;BN;;;;;N;;;;;````
- ````????;SENTENCE JOINER;Cf;0;BN;;;;;N;;;;;````

7. Official Proposal Summary Form

A. Administrative

1. Title
Proposal to encode 2E43 (SENTENCE NON-JOINER) and 2E44 (SENTENCE JOINER)

2. Requester's name
Arle Lommel (on behalf of the Unicode Localization Interoperability committee)

3. Requester type (Member body/Liaison/Individual contribution)
Individual contribution (on behalf of Unicode committee)

4. Submission date
2012-05-09

5. Requester's reference (if applicable)

6. Choose one of the following:
- This is a complete proposal
- ~~More information will be provided later~~

B. Technical – General

1. Choose one of the following:

1a. This proposal is for a new script (set of characters), Proposed name of script
No

1b. The proposal is for addition of character(s) to an existing block, Name of the existing block
Yes. SUPPLEMENTARY PUNCTUATION

2. Number of characters in proposal
2 (two)

3. Proposed category
[Unknown]

4. Is a repertoire including character names provided?
Yes

4a. If YES, are the names in accordance with the "character naming guidelines" in Annex L of P&P document?
Yes

4b. Are the character shapes attached in a legible form suitable for review?
Yes

5. Fonts related:

a. Who will provide the appropriate computerized font to the Project Editor of 10646 for publishing the standard?
Arle Lommel

b. Identify the party granting a license for use of the font by the editors (include address, e-mail etc.)
Arle Lommel (arle.lommel@gmail.com)

6a. Are references (to other character sets, dictionaries, descriptive texts etc.) provided?
No (not relevant)

6b. Are published examples of use (such as samples from newspapers, magazines, or other sources) of proposed characters attached?
No (not relevant since these serve as control-type characters)

7. Does the proposal address other aspects of character data processing (if applicable) such as input, presentation, sorting, searching, indexing, transliteration etc. (if yes please enclose information)?
Yes. These characters are proposed to assist in plain-text marking of sentence-level segment boundaries and disambiguation of potential break points generated by UAX #29.

8. Submitters are invited to provide any additional information about Properties of the proposed Character(s) or Script that will assist in correct understanding of and correct linguistic processing of the proposed character(s) or script.
See the detailed proposal

C. Technical – Justification

1. Has this proposal for addition of character(s) been submitted before? If YES, explain.
No

2a. Has contact been made to members of the user community (for example: National Body, user groups of the script or characters, other experts, etc.)?
Yes

2b. If YES, with whom?
Members of the ULI committee

2c. If YES, available relevant documents
This proposal has been discussed and contributed to by the members of the ULI committee

3. Information on the user community for the proposed characters (for example: size, demographics, information technology use, or publishing use) is included?
Users of automated translation tools, authoring tools, Internet search engines, content management tools and any other text-processing tools that might need to indicate how they segment text.

4a. The context of use for the proposed characters (type of use; common or rare)
Unknown, but potentially common. These character address an emerging need in text processing and will be required for ULI contributions to CLDR.

4b. Reference
See detailed proposal.

5a. Are the proposed characters in current use by the user community?
No. These represent a novel usage scenario needed to support ULI activities. There are markup- based equivalents, but none are universally accepted or usable with plain-text environments. (Internal testing with substitute characters has been carried out, however.)

5b. If YES, where?

6a. After giving due considerations to the principles in the P&P document must the proposed characters be entirely in the BMP?
Yes.

6b. If YES, is a rationale provided?
There are only two characters and given that their function and behavior is similar to existing characters, they should be encoded in the BMP with the analogous characters.

6c. If YES, reference

7. Should the proposed characters be kept together in a contiguous range (rather than being scattered)?
Ideally, yes, as these provide a functional pair and keeping them adjacent would offer mnemonic value

8a. Can any of the proposed characters be considered a presentation form of an existing character or character sequence?
No

8b. If YES, is a rationale for its inclusion provided?

8c. If YES, reference

9a. Can any of the proposed characters be encoded using a composed character sequence of either existing characters or other proposed characters?
No

9b. If YES, is a rationale for its inclusion provided?

9c. If YES, reference

10a. Can any of the proposed character(s) be considered to be similar (in appearance or function) to an existing character?
Yes

10b. If YES, is a rationale for its inclusion provided?
Yes

10c. If YES, reference
The similarity is to general classes of separators, but the specific semantics are distinct.

11a. Does the proposal include use of combining characters and/or use of composite sequences?
No

11b. If YES, is a rationale for such use provided?

11c. If YES, reference

11d. Is a list of composite sequences and their corresponding glyph images (graphic symbols) provided?

12a. Does the proposal contain characters with any special properties such as control function or similar semantics?
Yes

12b. If YES, describe in detail (include attachment if necessary)
See above.

13a. Does the proposal contain any Ideographic compatibility character(s)?
No

13b. If YES, is the equivalent corresponding unified ideographic character(s) identified?

13c. If YES, reference

8. Outstanding Issues
The following known issues need to be resolved prior to adoption of this proposal:

- The Unicode characters properties (Section 6) need to be verified. While we believe them to be so, these characters are unusual enough that more eyes would help.
- Section 6 of UAX #29 and the Sentence Break Property file will both need to be modified to allow for these characters. Successful modification will require assistance from expert parties.