

New International features of Internet Explorer

Michel Suignard

Microsoft Corporation

1 Summary

This document presents new implementations of international features by Microsoft Internet Explorer version 5 (5.0 and the forthcoming 5.5). Some are improvement of already existing features (font-linking, additional script support). Others are new and based on specifications presented to the W3C Internationalization Working Group and Style Working Group. The text in this presentation borrows in large parts from these specifications. As browsers are becoming more and more global viewers it is important to better support many East Asian features like Vertical writing (IE 5.5), Document Grid, Line Breaking (Kinsoku), Autospace and Ruby (phonetic annotation). IE5 has innovated in many of these areas. In the future we can expect that UAs (User Agents also commonly called browsers) will improve their support of the whole Unicode standard repertoire through additional presentation properties.

2 Contents

1	Summary	1
2	Contents	1
3	Font linking improvement	1
3.1	Script definition	2
4	Unicode 3.0 support	3
5	Surrogate support	3
6	Vertical writing	3
6.1	Relative versus physical.....	4
6.2	Auto-sizing.....	5
7	Document grid	5
8	Line breaking	6
8.1	Overview.....	6
8.2	CSS properties	7
8.3	Line breaking behavior table.....	8
9	Justification behaviors	9
10	Autospace	10
11	Ruby	11
11.1	Overview.....	11
11.2	Ruby presentation	12
11.2.1	Ruby box model	12
11.2.2	Ruby CSS properties	13
12	Conclusion	13

3 Font linking improvement

Font linking is a mechanism that allows text to be rendered even if the default font associated to a text element doesn't contain the necessary characters. It is basically a font substitution process that compares text content with the current font capability and selects a more appropriate font if necessary. Historically the mechanism has been character set based, i.e. the character set of a text sequence was compared to the character set rendering capability of the available fonts. New with IE5, the process now uses a script-based approach, which compares the script content of a text sequence with script rendering capability of the available fonts. It also uses other hints like an explicit language attribute to better determine the appropriate font.

The following is an example of what can be done with IE5 font linking:

“When the world wants to talk, it speaks Unicode”

عندما يريد العالم أن يتكلم، فهو يتحدث يونيكود

Երբ աշխարհը կուզէ խոսիլ կը գործածէ յունիքոս լեզուն
Munduak hitz egin nahi duenean, Unicodez hitz egiten du
Когда светът иска да общува, той говори на Unicode
Quan el món vol conversar, parla Unicode

当世界需要沟通时，请用 Unicode
當世界需要溝通時，請用統一碼 (Unicode)

Kad svijet želi razgovarati, govori Unicode
Když chce svět mluvit, mluví Unicode

Quand le monde veut communiquer, il parle en Unicode

როდესაც მსოფლიოს ურთიერთობა სურს იგი Unicode-ის ენაზე ლაპარაკობს

Wenn die Welt miteinander spricht, spricht sie Unicode

Όταν ο κόσμος θέλει να μιλήσει, μιλάει Unicode

כאשר העולם רוצה לדבר, הוא מדבר ב-Unicode

世界的に話すなら、Unicode です

세계를 향한 대화, 유니코드로 하십시오

Когда мир желает общаться, он общается на Unicode

Кад свет жели да разговара, говори Unicode

However font linking is a last resort mechanism that should not be used when exact rendering is preferred as the font selection is completely controlled by IE through the font selection dialog (Tools menu command). Furthermore as the matching between characters and appropriate fonts is done on a script basis there is not a 100% guarantee that the text will be fully rendered if the text contains characters that are rarely contained in fonts commonly supporting that script. In those cases it is preferable to use explicit font-family property values or the @font-face at-rules. These rules constitute a font description, typically a font-family descriptor and a location descriptor using an URI (ref W3C TR CSS2 4.1.5 and 15.3.1).

Finally when a script is used by writing systems with significant variant glyphs (like CJK) it is strongly suggested to mark the text with the language attribute, this provides a very useful about which font to select. By default, IE will use the charset meta-tag to better determine the appropriate font, i.e. an East Asian charset will provide an hint to which language is preferred.

3.1 Script definition

The Unicode space may be divided in the following script derived groups:

shown upright, other characters are typically shown rotated by 90 degree. The following figures show some example as isolated paragraphs and tables:

こ
こ
冬
は
寒
い
で
す
け
れ
ど
も
、
雪
あ
ま
り
降
り
ま
せ
ん
。

Vertical text (isolated paragraph)

<p>こ こ 冬 は 寒 い で す け れ ど も 、 雪 あ ま り 降 り ま せ ん 。</p> <p>Japanese です。</p>	<p>こ こ 冬 は 寒 い で す け れ ど も 、 雪 あ ま り 降 り ま せ ん 。</p> <p>こ こ 冬 は 寒 い で す け れ ど も 、 雪 あ ま り 降 り ま せ ん 。</p>
<p>こ こ 冬 は 寒 い で す け れ ど も 、 雪 あ ま り 降 り ま せ ん 。</p> <p>こ こ 冬 は 寒 い で す け れ ど も 、 雪 あ ま り 降 り ま せ ん 。</p>	<p>こ こ 冬 は 寒 い で す け れ ど も 、 雪 あ ま り 降 り ま せ ん 。</p> <p>こ こ 冬 は 寒 い で す け れ ど も 、 雪 あ ま り 降 り ま せ ん 。</p>

Vertical text in tables

6.1 Relative versus physical

When applied to HTML and CSS, the vertical writing creates some interesting challenges:

Vertical writing is by some aspect a text rotation of 90 degree, especially when looking at the non-ideographic text. Therefore, it could be tempting to apply to all HTML and CSS concepts a logical rotation. For example what looks like a logical top of the paragraph is really on the right side. This works reasonably well for concept like text alignment, line height, etc... but starts to be fairly confusing for more physical concept like width, height, top, bottom, etc...

In W3C a consensus emerged to treat most of the positioning model as a physical model (i.e. left is always on the left, even on a box containing vertical text), unless the name is really a misnomer. In the latter case, the concept is considered from a logical or relative point of view (that is, relative to the text and block orientation). Such cases are:

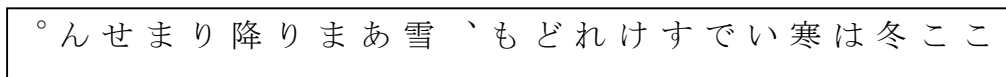
letter-spacing, line-height, text-align, text-indent, etc...

6.2 Auto-sizing

Another issue has to do with auto-sizing when writing modes are combined, like mixing cells with various writing directions in a table and inserting a vertical paragraph in an horizontal division.

Typically, sizing favors width over height, but for vertical writing the height (relative line width) is much more important. Again some considerations worth noting:

The minimum height of an ideographic text is not typically the shorter distance between two line-breaking opportunities (like it would be for Latin text). Doing so could result in vertical ideographic text that looks like right to left ideographic (one character per column), like in the following example:

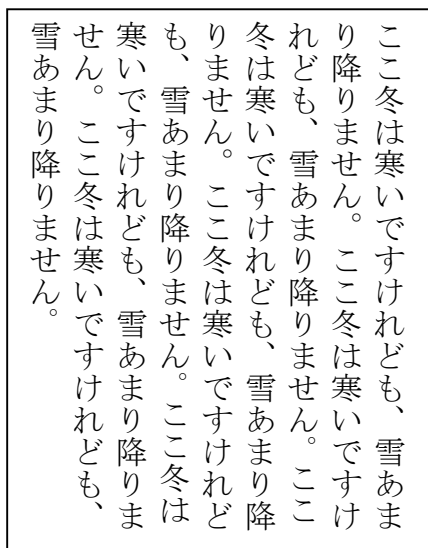


The second challenge is to optimally fill a vertical text container where the width is known, but the height is unknown. The optimal solution requires an iterating process that creates a performance issue.

IE5.5 solves these two issues as following:

1. Determine a minimum height for vertical text; it results in a minimum of about 10-12 characters that is typical in publication in East Asian writing systems using that mode.
2. Use two sizing attempts (one with a maximum height and one with a minimum height) to come up with an optimal height in as little iteration as possible.

The following box can be seen as the result of an optimal filling (height containing more than 10/12 characters, width totally filled):



To conclude, in many cases, especially cases involving mixes of writing direction it is highly desirable to author precisely width and height of the containers. It goes against some of the principle of HTML authoring, but despite all best efforts of browsers, the auto-sizing has its limitation.

7 Document grid

It is very common for the glyphs in documents written in East Asian languages, such as Chinese or Japanese, to be laid out on the page according to a specified one- or two-dimensional grid. The concept of grid can also be used in other, non-ideographic contexts such as Braille or monospaced layout.

The diagram below represents a fragment of horizontal text on a page with mixed wide-cell and narrow-cell glyphs that a Japanese user intended to be laid out on a grid that resulted in 9 glyphs per line (gray grid lines shown for clarity):

こ	れ	は	日	本	語	の	文	書
で	す	。	This is an English					
sentence								

Figure 1: 'Genko' grid applied to mixed text

The grid affects not only the placement of the glyphs, but it can also modify the behavior of several other layout-related behaviors, such as indent size, margins or paragraph alignment.

One can distinguish between three types of grid: a *strict* one, used mostly in Chinese, but also occasionally in Japanese (a.k.a. "genko"), a *loose* one, frequently used in Japanese and sometimes in Korean, as well as a *fixed* one, potentially useful for non-ideographic text, such as Braille or mono-spaced layout in general.

Strict applies a true grid mode to each 'wide' character (one character per grid cell or per multiple of grid cells if the character cannot fit into a single cell); others are merged in character strips that are then treated similarly to a single 'wide' character. The loose value can be seen as applying a constant width increment to each character width (although the increment is different between 'wide' and 'narrow' characters). Fix applies a true grid mode to all characters (except connected characters).

The grid type entails a set of layout rules that determine how much flexibility the UA is allowed to have when laying out a line of text.

Different grids can be defined for different parts of the document.

The grid can be selectively disabled in either dimension on fragments of text.

Line grid can be disabled for individual paragraphs. If line grid is disabled for a paragraph, the lines of the paragraph are laid out just as if no line grid were specified. The glyphs in a paragraph with line grid disabled still follow the character grid, if one is specified.

Accordingly, the grid behavior is determined through the following properties:

- layout-grid-type determines the grid type: loose, strict or fixed.
- layout-grid-mode determines the grid mode: none, line, char or both.
- layout-grid-line determines the line grid value.
- layout-grid-char determines the grid char value.

There is also a shorthand notation to combine these properties (called layout-grid).

8 Line breaking

8.1 Overview

Line breaking is the process by which line breaking opportunities are determined in a given line of text. In Western text, those opportunities are based either on 'space/white space' or hyphenation. In many East Asian writing systems, line-breaking opportunities exist almost anywhere. The forbidden cases are specified as 'Kinsoku' in Japanese and can be extrapolated to a Chinese context. For example it is not desirable to have a closing parenthesis ')' starting a line. Korean rules are somewhere between Western and Japanese rules as Korean texts typically use space as a word delimiter. Determining line breaking opportunities for Asian writing systems usually requires a two character

context, possibly three if the decision to whether breaking across space or not is also analyzed. (It may be seem obvious that we should always break across space, but there are cases in Japanese typography where it is not desirable to do so).

The following context elements should be considered:

- Provide normal Kinsoku rules. This is the standard mode where symbols are allowed to either start or end a line based on their relationship to previous or following characters. In that case most ideographic characters are allowed to either start or end a line.
- Apply stricter Kinsoku rules by not allowing small kana characters to start a line.
- Thai line breaking is performed using a contextual dictionary lookup. This specification assumes that a special module provides the following information: Last character of a Thai word, First character of a Thai Word, other Thai letter.

Two additional contextual elements apply to the previous rules:

- Allow Western text (really non-East Asian text) to break everywhere. This may provide incorrect line breaking within Western text but is more visually pleasing in Asian context.
- Keep ideographic characters and Hangul together. This mode is used in Korean context where typically Hanja and Hangul appear in clusters separated by space characters.

8.2 CSS properties

The line-break property selects the line breaking level. Possible values:

- *normal* - line breaking opportunities are determined by the Latin, Thai and regular ideographic line breaking rules with an additional commonly used set of restrictions.
- *strict* - line breaking opportunities are determined by the Latin, Thai and regular ideographic line breaking rules with an additional larger set of restrictions.

The word-break property controls line-breaking behavior inside of words. Possible values:

- *normal*: Keep all non Asian scripts (according to their own rules), CJK and Hangul break everywhere or according to the rules of line-break mode (kinsoku).
- *break-all*: same as normal for CJK and Hangul, but now non Asian scripts can also break anywhere. This options is used mostly in context where the text is predominantly using CJK characters with few non Asian excerpts and it desired that the text be better distributed on each line.
- *keep-all*: same as normal for all non Asian scripts. CJK and Hangul are kept together. This option should only be used in the context of CJK used in small clusters like in the Korean writing system.

For convenience, the word-break property can also expressed using two boolean flags combining word-break values:

- *break-latin*
 - *on* when word-break = break-all
 - *off* when word-break = normal | keep-all
- *break-CJK*
 - *on* when word-break = normal | break-all
 - *off* when word-break = keep-all

The line breaking behavior following table uses these two flags directly.

8.3 Line breaking behavior table

The following table determines for every character pairs (character 'Before' and character 'After') classified in line breaking classes (21 in this example) the line breaking opportunities. Line breaking classes are determined by applying a unique classification to each character from the Unicode repertoire. The determination can be contextual (mostly character 'width': narrow, wide, ambiguous, etc.). There are five (0 to 4) kinds of opportunities used by the table-based line breaking logic.

<u>Before character</u>	Class#	<u>After character</u>																				
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Opening characters	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	1
Closing characters	2	0	1	1	1	0	0	2	0	0	0	0	2	1	2	1	0	0	0	0	2	1
No start ideographic	3	0	1	2	1	2	0	2	4	0	0	4	2	1	2	1	4	0	0	0	2	1
Exclamation/interrogation	4	0	1	2	1	0	0	0	0	0	0	0	2	1	2	1	0	0	0	0	2	1
Inseparable	5	0	1	2	1	2	0	0	0	0	0	0	2	1	2	1	0	0	0	0	2	1
Prefix	6	2	1	2	1	0	2	0	2	2	0	3	2	1	2	1	2	2	0	0	2	1
Postfix	7	0	1	2	1	0	0	0	0	0	0	0	2	1	2	1	0	0	0	0	2	1
Ideographic	8	0	1	2	1	2	0	2	4	0	0	4	2	1	2	1	4	0	0	0	2	1
Numeral sequence	9	0	1	2	1	2	0	2	0	3	0	3	2	1	2	1	0	0	0	0	2	1
Alpha space	10	0	1	2	1	0	0	0	0	0	0	0	2	1	2	1	0	0	0	0	2	1
Alpha characters/symbols	11	0	1	2	1	2	2	2	4	3	0	3	2	1	2	1	4	0	0	0	2	1
Glue Characters	12	2	1	2	1	2	2	2	2	2	2	2	2	1	2	1	2	2	2	2	2	1
Slash	13	0	1	2	1	2	0	2	0	2	0	3	2	1	2	1	0	0	0	0	2	1
Quotation characters	14	1	1	2	1	2	2	3	2	2	2	2	2	1	2	1	2	2	2	2	2	1
Numeric separators	15	0	1	2	1	0	2	2	0	2	0	3	2	1	2	2	0	0	0	0	2	1
Hangul	16	0	1	2	1	2	0	2	4	0	0	4	2	1	2	1	4	0	0	0	2	1
Thai first	17	0	1	2	1	2	0	2	0	0	0	0	2	1	2	1	4	2	2	2	2	1
Thai last	18	0	1	2	1	2	0	2	0	0	0	0	2	1	2	1	0	0	2	2	2	1
Thai middle	19	0	1	2	1	2	0	0	0	0	0	0	2	2	2	1	0	2	2	2	2	1
Combining	20	0	1	2	1	2	2	2	4	3	0	3	2	1	2	1	4	0	0	0	1	1
Ascii space	21	0	1	2	1	0	0	0	0	0	0	0	2	1	2	1	0	0	0	0	2	1

Line Break Behavior table

Notes:

- 0 indicates that line breaking is allowed between these 2 classes.
- 1 indicates no line breaking as well as no break across space,
- 2 indicates no line breaking, but can break across space.
- 3 is like 0 (lb) when break-latin is on, otherwise it is 2,

- 4 is like 0 (lb) when break-CJK is on, otherwise it is 2.
- The Ascii Space column and row are typically inactive, i.e. the space character is only processed as part of the algorithm of breaking across 'space'. There is however few cases where it is desirable to process the space character as a standard character (TEXTAREA), in those cases these columns and rows are used.

Thai line breaking

The Thai language is one of several languages that do not require white space between words. And it is not possible through simple character classification to determine line breaking. Therefore for Thai, the context around the character needs to be analyzed to determine breaking opportunities. Once done, the result is fed back into the Kinsoku logic to avoid separate path and also to make easier to process boundary cases. The following example shows how the logic is implemented:

Assuming that in a string called 'LabcdefM', 'L' and 'M' are Latin characters, 'abcded' are Thai characters with a breaking opportunity between 'c' and 'd', two breaking classes are provided for each character:

	brkclsAfter	brkclsBefore
L	Alpha	Alpha
a	ThaiMiddle	ThaiFirst
b	ThaiMiddle	ThaiMiddle
c	ThaiLast	ThaiMiddle
d	ThaiMiddle	ThaiFirst
e	ThaiMiddle	ThaiMiddle
f	ThaiLast	ThaiMiddle
M	Alpha	Alpha

And the following classes are used for the breaking opportunity analysis as input in the line breaking behavior table:

	First character (before)	Second character (After)
{L, a}	Alpha	ThaiFirst
{a, b}	ThaiMiddle	ThaiMiddle
{b, c}	ThaiMiddle	ThaiMiddle
{c,d}	ThaiLast	ThaiFirst
{d, e}	ThaiMiddle	ThaiMiddle
{e, f}	ThaiMiddle	ThaiMiddle
{f, M}	ThaiLast	Alpha

It should be noted that with this logic, the line 17 (Thai first) and the column 18 (Thai Last) should never be used..

9 Justification behaviors

Until now, there was only one line justification mode in HTML/CSS. Setting the CSS property *'text-align'* to justify activated it. IE5 adds a new level of justification modes by using a new proposed CSS property named *'text-justify'* that can takes the following values (in addition to *auto*):

- **inter-word.** This is the simplest line justification as it only affects inter word spacing by increasing the width of the space between these words. In this mode the last line is not justified (flush left). No expansion or compression occurs within words.

- **newspaper.** This mode allows expansion and compression to take place at both inter words and intra words. This mode is especially useful for narrow columns. Typically, compression is tried first, if unsuccessful, expansion occurs: inter word spaces are expanded up to a threshold, and finally inter letter expansion is performed. The threshold value is related to the column width (in number of characters). In this mode the last line is not justified (flush left).
- **distribute.** This mode is related to newspaper as it also involves compression and expansion. It is however targeted at East Asian writing systems as it distributes evenly width-increases between letters and relies less on the presence of space characters and related expansion thresholds. In this mode the last line is not justified (flush left).
- **inter-ideograph.** This mode has the same scope as the distribute modes. It however restricts significantly the number of cases where expansion can be performed. Basically inter-letter expansion is reserved to ideographic characters. This is the preferred justification in the context of East Asian writing systems. In this mode the last line is not justified.

inter-ideograph justification

This is a text with mixed English and Kanji Text.
東京で夏は無視圧かった。
私の日本語だめです。英語
で日本は Japanese です。
Some English text 全然分
かりま 12345678 せんでし
た。

distribute-all-lines justification

This is a text with
mixed English and Kanji
Text. 東京で夏は無視圧か
った。私の日本語だめで
す。英語で日本は Japanese
です。Some English text
全然分かりま 12345678 せ
ん で し た 。

10 Autospace

When a run of non-ideographic or numeric characters appears inside of ideographic text, a certain amount of space is often preferred on both sides of the non-ideographic text to separate it from the surrounding ideographic glyphs. The *text-autospace* property controls the creation of that space when rendering the text. That added width does not correspond to the insertion of additional space characters, but instead to the width increment of existing glyphs. (A commonly used algorithm for determining this behavior is specified in JIS X-4051.)

This property is additive with the *word-spacing* and *letter-spacing* CSS2 properties, that is, the amount of spacing contributed by the '*letter-spacing*' setting (if any) is added to the spacing created by '*text-autospace*'. The same applies to '*word-spacing*'. Possible values:

- **none** - no extra space is created.
- **ideograph-numeric** - creates extra spacing between runs of ideographic text and numeric glyphs.
- **ideograph-alpha** - creates extra spacing between runs of ideographic text and non-ideographic text, such as Latin-based, Cyrillic, Greek, Arabic or Hebrew.
- **ideograph-space** - extends the width of the space character while surrounded by ideographs.
- **ideograph-parenthesis** - creates extra spacing between normal (non wide) parenthesis and ideographs.

Here are some examples of these effects.

No autospace

英語 で[日本]はJapaneseです。全然123分かりませんでした。

Autospace with numeric only

英語 で[日本]はJapaneseです。全然 123 分かりませんでした。

Autospace with alpha only

英語 で[日本]は Japanese です。全然123分かりませんでした。

11 Ruby

11.1 Overview

The "ruby" is the commonly used name for a run of text that appears in the immediate vicinity of another run of text, referred to as the "base", and serves as an annotation or a pronunciation guide associated with that run of text. Ruby, as used in Japanese, is described in JIS-X-4051.

The font size of ruby text is normally half the font size of the base. The name "ruby" in fact originated from the name of the 5.5pt font size in British printing, which is about half the 10pt font size commonly used for normal text.

There are several positions where the ruby text can appear relative to its base.

Ruby text normally appears alongside the base. It is most frequently placed above the base in horizontal layout.

に ほ ん ご ← *Ruby text*
日本語 ← *Ruby base*

In certain scenarios, however, ruby text may appear as inline text immediately following the base. The inline form serves only as a fallback for limited resolution display media or older software.

When the ruby text appears inline, it is enclosed within parentheses. The parentheses however are used only inline.

日本語 (にほんご)

Note however, that using parentheses for the fallback may lead to confusion between runs of text intended to be ruby text and others that happen to be enclosed within parentheses. The author should be aware of the potential for that confusion and is advised to choose an unambiguous delimiter for the fallback, if this is a concern.

This document introduces a ruby model in HTML using the new ruby element serving as the container for the rb, rt and rp elements. The rb element contains the base characters of the ruby. The rt contains the ruby text and the rp elements contains the parenthesis characters used in the fallback case. The rb element may contain another ruby element. The author can achieve the case of ruby text appearing both below and above the same base by nesting a ruby element inside the rb element of another ruby. For example, the following ruby:

World Wide Web
W W W

can be represented using the following markup in XML-ized HTML [[XHTML](#)], which is the full form:

```
<ruby><rb>WWW</rb><rp>(</rp><rt>World Wide Web</rt><rp>)</rp></ruby>
```

or using the following markup in SGML HTML, where it is possible to omit the end tags of rb, rt and rp:

```
<ruby>WWW<rp>(<rt>World Wide Web<rp>)</ruby>
```

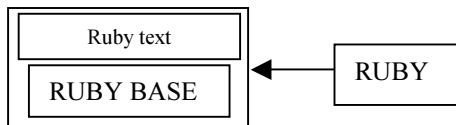
If the author is not concerned about displaying ruby inside of parentheses in older UA's or he is working in generic XML environment that supports CSS2 style sheets, he may skip the rp element altogether:

```
<ruby><rb>WWW</rb><rt>World Wide Web</rt></ruby>
```

11.2 Ruby presentation

11.2.1 Ruby box model

In a UA that supports ruby, the ruby structure consists of three boxes. The outermost container is the ruby element itself. It is a container for two non-overlapping boxes: the ruby text box and the ruby base box. The positioning of these two boxes relative to each other is controlled by '*ruby-position*'.



The width of the ruby box is by default determined by its widest child element, whose width in turn is determined by its content. Both of ruby's children assume the width of the widest one of them. In this respect, the ruby box is much like a two-cell table element, with the following exceptions:

- the ruby box is an inline element, like an image, even though it itself, like a table, is a container of (two) other boxes
- both the ruby text and the ruby base boxes may overlap with adjacent text if an appropriate '*ruby-overhang*' parameter is set via CSS.

If the ruby text is not allowed to overhang anything, then the ruby behaves like a traditional box, i.e. only its contents are rendered within its boundaries and adjacent elements do not cross the box boundary. However, if ruby text is allowed to overhang adjacent elements and it happens to be wider than its base, then the adjacent content is partially rendered within the area of the ruby base box, while the ruby text may be partially overlapping with the upper blank parts of the adjacent content:

The Ruby text from one base can never overhang another ruby base.

The alignment of the contents of the base or the ruby text is not affected by the overhanging behavior. The alignment is achieved the same way regardless of the overhang behavior setting and it is computed before the space available for overlap is determined. It is controlled by the '*ruby-align*' property.

The exact circumstances in which the ruby text will overhang other elements, and to what degree it will do so, will be controlled by ruby CSS properties.

11.2.2 Ruby CSS properties

The following properties are available:

- *ruby-position* (value: above | below | inline). This property is used on the ruby element to control the position of the ruby text with respect to its base.
- *ruby-align* (value: auto | left | center | right | distribute-letter | distribute-space | line-edge). This property can be used on any element to control the text alignment of the ruby text and ruby base contents relative to each other. It applies to all the ruby's in the element. The alignment is applied to the ruby child element whose content is shorter: either the rb or the rt element. Possible values:
 - auto - The browser determines how the ruby contents are aligned.
 - left - The ruby text contents are left aligned with the base.
 - center - The ruby text contents are centered within the width of the base. If the length of the base is smaller than the length of the ruby text, then the base is centered within the width of the ruby text.
 - right - The ruby text contents are right aligned with the base.
 - distribute-letter - If the width of the ruby text is smaller than that of the base, then the ruby text contents are evenly distributed across the width of the base, with the first and last ruby text glyphs lining up with the corresponding first and last base glyphs. If the width of the ruby text is at least the width of the base, then the letters of the base are evenly distributed across the width of the ruby text. This type of alignment is sometimes referred to as the "0:1:0" alignment.
 - distribute-space - If the width of the ruby text is smaller than that of the base, then the ruby text contents are evenly distributed across the width of the base, with a certain amount of white space preceding the first and following the last character in the ruby text. That amount of white space is normally equal to half the amount of inter-character space of the ruby text. If the width of the ruby text is at least the width of the base, then the same type of space distribution applies to the base. In other words, if the base is shorter than the ruby text, the base is distribute-space aligned. This type of alignment is sometimes referred to as the "1:2:1" alignment.
 - line-edge - If the ruby text is not adjacent to a line edge, it is aligned as in 'auto'. If it is adjacent to a line edge, then it is still aligned as in auto, but the side of the ruby text that touches the end of the line is lined up with the corresponding edge of the base. This type of alignment is relevant only to the scenario where the ruby text is longer than the ruby base. In the other scenarios, this is just 'auto'.
- *ruby-overhang* (value: auto | start | end | none). This property determines whether, and on which side, ruby text is allowed to partially overhang any adjacent text in addition to its own base, when the ruby text is wider than the ruby base. Note that ruby text is never allowed to overhang glyphs belonging to another ruby base.

12 Conclusion

Supporting complex constructs like the Ruby allows Internet Explorer to improve its coverage of the East Asian market. But there is still much to do. Many new properties like layout-flow, emphasis mark, hanging punctuation are being specified and will be implemented in future versions. Along with support for additional scripts we can expect browsers to become truly **word** wide web tools.