CSS3 Text Extensions

Michel Suignard

Microsoft Corporation

1 Summary

This document presents new text extensions considered for CSS3 (Cascading Style Sheet). The main topics presented are layout flow, text justification, baseline alignment, line breaking, document grid and Ruby (phonetic annotation). These features benefits greatly from the Unicode Standard and its related technical reports, which specify several character properties that are essential for the specification and implementation of these typographical features. The paper is based on a W3C working draft (CSS3 text module) available to W3C member companies.

2 Contents

3 Text Layout

This section describes the text layout features supported by CSS3, which includes support for various international writing directions, such as left-to-right (e.g., Roman scripts), right-to-left (e.g., Hebrew or Arabic), bi-directional (e.g., mixing Roman with Arabic) and vertical (e.g., Asian scripts).

The writing-mode property determines an inline progression and a line-to-line progression, also called block progression. For example, Roman scripts are typically written left to right and top to bottom. The glyph orientation determines the orientation of the rendered visual shape of characters relative to the primary text advance direction.

Bi-directionality introduces another level of complexity in text layout, as in many combinations of writing-mode and glyph orientation values the proper directionality of text will be determined by an algorithm. The Unicode standard (section 3.12) defines such an algorithm consisting of an implicit part based on character properties, as well as explicit controls for embeddings and overrides.

3.1 Setting the primary text advance direction: the 'writing-mode' and 'direction' properties

The 'writing-mode' property specifies the following text layout flow:

- **lr-tb**, Left to right, top to bottom, used by Latin-based writing systems,
- **rl-tb**, Right to left, top to bottom, used by Hebrew and Arabic,
- **tb-rl**, Top to bottom, right to left, used in Japan and China, in addition to the lr-tb mode,
- **tb-lr**, Top to bottom, left to right, Mongolian, Western signage,
- **bt-lr**, Bottom to top, left to right, historic (Hanunoo in the Philippines)
- **bt-rl**, Bottom to top, right to left, no known usage, required for consistency

An example of vertical text (tb-rl):

ここ冬は寒いで
すけれども、雪
あまり降りませ
ん。ここ冬は寒
いですけれど
も、雪あまり降
りません。

The ‘direction’ property specifies the line direction component of the text advance direction and the direction of embeddings and overrides (see 'unicode-bidi') for the Unicode bi-directional algorithm. It takes the following values:

- **ltr** (left to right direction),
- **rtl** (right to left direction).

The usage of the 'direction' property for block-level elements is discouraged in CSS3 as the 'writing-mode' property supersedes it.

3.2 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…

3.3 Glyph orientation within a text run: the 'glyph-orientation-vertical' and 'glyph-orientation-horizontal' properties

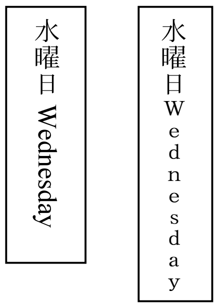
In some cases, it is required to alter the orientation of a sequence of characters relative to the primary text advance direction. The requirement is particularly applicable to vertical layouts of East Asian documents, where sometimes half-width Roman text is to be displayed horizontally and other times vertically.

Two properties control the glyph orientation relative to the primary text advance direction: 'glyph-orientation-vertical' controls glyph orientation when the primary text advance direction is vertical; 'glyph-orientation-horizontal' controls glyph orientation when the primary text advance direction is horizontal. The ‘glyph-orientation-vertical’ takes the following values:

- **<angle>**, typically an angle of 0, 90, 180 or 270 degree
- **auto**, describe character behavior as dictated by font and typical OS usage (for example ideographic characters are upright, Latin characters on the ‘side’).

The ‘glyph-orientation-horizontal’ only takes the ‘**<angle>**’ values. The value ‘auto’ is not supported because there is no typical usage of mixed glyph orientation in horizontal flow.

The two following text shows two cases of glyph-orientation vertical, the first being ‘auto’, the second ‘0deg’:



3.4 Embedding and override: the 'unicode-bidi' property

This property allows further control of the Unicode bidirectional algorithm by allowing new embedding level or direction override. Values for this property have the following meanings:

- **normal**, the element does not open an additional level of embedding with respect to the bidirectional algorithm. For inline-level elements, implicit reordering works across element boundaries.
- **embed**, if the element is inline-level, this value opens an additional level of embedding with respect to the bidirectional algorithm. The direction of this embedding level is given by the 'direction' property. Inside the element, reordering is done implicitly. This corresponds to adding a LRE (U+202A; for 'direction: ltr') or RLE (U+202B; for 'direction: rtl') at the start of the element and a PDF (U+202C) at the end of the element.
- **bidi-override**, if the element is inline-level or a block-level element that contains only inline-level elements, this creates an override. This means that inside the element, reordering is strictly in sequence according to the 'direction' property; the implicit part of the bidirectional algorithm is ignored. This corresponds to adding a LRO (U+202D; for 'direction: ltr') or RLO (U+202E; for 'direction: rtl') at the start of the element and a PDF (U+202C) at the end of the element.

4 Script character classification: the 'script' property

In text layout, many of the behaviors are related to a character classification based on scripts. For example, line breaking or text justification behaviors depend on the 'dominant' script of the textual content of an element. This can be heuristically determined by finding the first character that has an unambiguous script identifier in an element. Using the ‘script’ property can also explicitly specify it. The property can take the following values:

- **auto**, Use the first character descendant[, after any re-ordering due to character direction and bi-directionality,] which has an unambiguous script identifier to determine the dominant script of the element's content. This determines the computed script value. Each textual component of the element may however behave in typographical related behaviors as dictated by its script identifier. In the absence of any textual components with a clear script identifier (or no textual content at all), the computed value is 'Latin'.
- **none**, Indicates the script is unknown or is not significant to the proper formatting of this element.
- **<script>**, a script definition in conformance with [ISO15924]. All textual components of the element must behave in typography related behaviors as dictated by this script value, not the inherent script value of these textual components.

5 Text Justification

5.1 The text-justify property

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:

- **auto**. The user agent (commonly called ‘browser’) determines the justification algorithm to follow, based on a balance between performance and adequate presentation quality. Inter-word expansion is typically used for all scripts that use space as word delimiter. If the ‘text-kashida-space’ property has a non zero-value, it is recommended to use kashida elongation for Arabic text.

- **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).

- **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.

- **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.
- **inter-cluster**. Plays the same role as inter-ideograph but for South Eastern Asian scripts like Thai, Khmer and Lao. A cluster is defined as a group of characters belonging to those scripts shaped as a single unit.
- **kashida**. Plays the same role as inter-ideographs but for Arabic through the Kashida effect.

The following table describes the expansion/compression strategy for the combination of each script groups and the text-justify property value for each relevant text-justify property value:

	text-justify property value						
Script groups	**auto***	**inter-word**	**newspaper**	**inter-ideograph**	**distribute**	**inter-cluster**	**kashida**
Latin	word-spacing only*	word-spacing only	prioritization between word-spacing and letter-spacing	word-spacing only	word-spacing only	word-spacing only	word-spacing only
CJK	no extra spacing*	no extra spacing	letter-spacing	letter-spacing	letter-spacing	no extra spacing	no extra spacing
Devanagari*	word-spacing*	word-spacing	word-spacing	word-spacing	word-spacing	word-spacing	word-spacing
Arabic	kashida and word-spacing*	word-spacing	kashida and word-spacing	word-spacing	kashida and word-spacing	word-spacing	kashida and word-spacing
SE Asian clusters	inter-cluster spacing*	inter-cluster spacing	inter-cluster spacing	no extra spacing	inter-cluster spacing	inter-cluster spacing	no extra spacing

*The values shown for the auto column are only a recommendation. The user agents might implement a different strategy.

*The Devanagari entry represents as well other scripts and writing systems used in India that use baseline connectors like Bengali and Gurmukhi.

5.2 The text-align-last property

This property describes how the last line of the inline content of a block is aligned. This also applies to the only line of a block if it contains a single line, the line preceding a BR element and to last lines of anonymous blocks. Typically the last line is aligned like the other lines of the block element, this is set by the 'text-align' property. However, in some situations like when the 'text-align' property is set to 'justify', the last line may be aligned differently. The property takes the following values:

- **auto**, The last line will be aligned like the other lines, i.e., determined by the value of the 'text-align' property. However, if the 'text-align' property is set to the value 'justify', the last line will be aligned to the start of the inline progression.
- **start**, **end** and **center** text respectively.

- **justify**, The last line will be justified like the other lines, using the justification type set by the 'text-justify' property. Note however that if there is no expansion opportunity in the last line, the line might not appeared justified.
- **size**, The line content is scaled to fit on the line. All the fonts on the line must be scaled by the same factor, and that factor must be as small as possible (i.e. fit as much on a line as possible). Typically this value is used for single line element.

Two additional properties: ‘min-font-size’ and ‘max-font-size’ allow the fonts used in the last line of an element to be adjusted between a minimum and a maximum font size. The properties are described in the CSS3 proposal.

The following text shows 2 examples of justification with different last line setting (‘auto’ and ‘justify’)

inter-ideograph justification with last line aligned to the left.

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 justification with last line justified.

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 。

5.3 The text-kashida property

Kashida is a typographic effect used in Arabic writing systems that allows character elongation at some carefully chosen points in Arabic. Each elongation can be accomplished using a number of kashida glyphs, a single graphic or character elongation on each side of the kashida point. (The user agent may use either mechanism based on font or system capability). The text-kashida-space property expresses the ratio of the kashida expansion size to the white space expansion size, 0% means no kashida expansion, 100% means kashida expansion only . This property can be used with any justification style where kashida expansion is used (currently text-justify: auto, kashida, distribute and newspaper).

The ‘text-kashida’ property takes a <percentage> as value to express the ration specified above. For example, the following text uses a ratio of 100%, meaning that only kashida expansion occurs (note however that the last line is not affected).

عـــــندما يـــــريدالعالم أن يـــــتكلّم، فهـــــو
يتحدّث يونيكود

6 Baseline alignment

Baseline alignment is the process by which various inline and inline-block elements are aligned between themselves. At the inner level, each glyph has an inherent alignment point which is the position that connects it to the previous visual element (typically another glyph). That alignment point is associated with an alignment baseline that is specific to the script to which the glyph belongs (examples are alphabetic, ideographic, hanging, etc.). In addition, the alignment is controlled by a dominant baseline, defining a priority among the script based baseline and a scale to adjust the various baselines which is typically independent of the size of the element being aligned.

There are four properties that control alignment of elements: 'dominant-baseline', 'alignment-baseline', 'baseline-shift' and 'alignment-adjust'. These properties are independent and are designed so that typically only the specification of one of the properties is needed to achieve a particular alignment goal.

The primary baseline alignment property is the 'dominant-baseline' property. This property has a compound value with three components:

1. The dominant-baseline-identifier component is the default 'alignment-baseline' to be used when aligning two inline areas.
2. The baseline-table component specifies the positions of the baselines in the font design space coordinates. The baseline-table acts something like a musical staff; it defines particular points along the block-progression-direction to which glyphs and inline elements can be aligned.
3. The baseline-table 'font-size' component provides a scaling factor for the baseline-table.

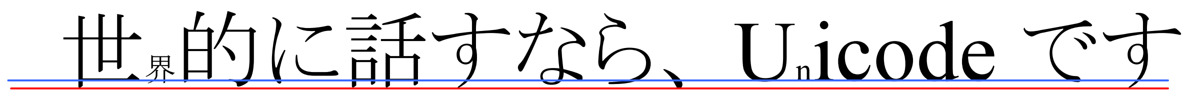
The second baseline alignment property is the ‘alignment-baseline’ property. The model assumes that each glyph has an inherent alignment-baseline value that specifies the baseline with which the glyph is to be aligned. (The 'alignment-baseline' is called the "Baseline Tag" in the OpenType baseline-table description.) The initial value of the 'alignment-baseline' property uses the baseline identifier associated with the given glyph. Alternate values for 'alignment-baseline' can be useful for glyphs such as a "*" which are ambiguous with respect to script membership.

The third baseline alignment property is the 'baseline-shift' property. Like the properties other than the "dominant-baseline" property, this property does not change the baseline-table or the baseline-table font-size. It does shift the whole baseline table of the parent element so that when an inner inline element is aligned to one of the parents baselines, the position of the inner inline element is shifted.

The fourth alignment property is the 'alignment-adjust' property. This property is primarily used for elements, such as some graphics, that do not belong to a particular script and do not have a predefined alignment point. The "alignment-adjust" property allows the author to assign where, on the start-edge of the object, the alignment point for that element lies.

These properties are described in details in the CSS3 text specification as well as the latest XSL Candidate Recommendation.

The following text shows a simple case of script based alignment where the ideographic characters are aligned slightly lower (red line) than the alphabetic characters (blue line). The size of two characters (corresponding to the 2nd character of each script run) has been lowered to show the alignment effect.



7 Line breaking

7.1 Overview

In documents written in Latin-based languages, where runs of characters make up words and words are separated by spaces or hyphens, line breaking is relatively simple. In the most general case, (assuming no hyphenation dictionary is available to the user agent), a line break can occur only at white-space characters or hyphens, including U+ 00AD SOFT HYPHEN.

In ideographic typography, however, where what appears as a single glyph can represent an entire word and neither spaces nor any other word separating characters are needed, a line breaking opportunity is not as obvious as a space. It can occur after or before many other characters. Certain line breaking restrictions still apply, but they are not as strict as they are in Latin typography.

Thai is another interesting example with its own special line breaking rules. Since Thai words are made up of runs of characters, it resembles Latin in that respect. But the lack of spaces as word delimiters, or in fact any consistent word delimiters, makes it similar to CJK. Thai, like Latin in the absence of a hyphenating dictionary, never breaks inside of words. In fact, knowledge of the vocabulary is necessary to be able to correctly break a line of Thai text. In addition, the Unicode character: U+200B ZERO WIDTH SPACE can be inserted in such scripts to specify an explicit line breaking opportunity.

A number of levels of line-breaking "strictness" can be used in Japanese typography. These levels add or remove line-breaking restrictions. The model presented in this specification distinguishes between two most commonly used line breaking levels for Japanese text, using the 'line-break' property.

In ideographic typography, it is also possible, though not always preferred, to allow line breaks to occur inside of quoted Latin and Hangul (Korean) words without following the line breaking rules of those particular scripts. The model proposed in this document gives the author control over that behavior through the 'word-break-CJK' property.

Finally, additional word breaking opportunities are controlled by 'word-break-wrap' and 'word-break-inside', allowing emergency word breaking for long words and hyphenation. All these properties are also available through the 'word-break' short hand property.

The Unicode Technical Report (TR#14), available from the Unicode Web site, also covers this subject. It contains a detailed recommendation and corresponding data for each Unicode character.

7.2 The ‘line-break’ property

This property selects the set of line breaking rules to be used for text. The values described below are especially useful to CJK authors, but the property itself is open to other, not yet specified settings for non-CJK authors as well. The property takes the following values:

- **normal**, Selects the normal line breaking mode for CJK. While the user agent is free to define its own line breaking restrictions for the 'normal' mode, it is recommended that breaks between small katakana and hiragana characters be allowed. That is the preference in modern Japanese typography, and is especially desirable for narrow columns. Japanese katakana words tend to be long, and it is preferable to allow line breaks to occur among such characters than to have excessive expansion due to justification.
- **strict**, Selects a more restrictive line breaking mode for CJK text. While the user agent is free to define its own line breaking restrictions for the 'strict' mode, it is recommended that the restrictions specified by the JIS X-4051 be followed. That implies that in this mode, small katakana and hiragana characters are not allowed to start a line.

In Japanese, a set of line breaking restrictions is referred to as "Kinsoku". JIS X-4051 is a popular source of reference for this behavior using the *strict* set of rules. This architecture involves character classification into line breaking behavior classes. Those classes are then analyzed in a two-dimensional behavior table where each row-column position represents a pair action to be taken at the occurrence of these classes. For example, given a closing character class and an opening character class, the intersection in that table of these two classes (the first character belonging to the opening class and the second belonging to the closing class) will indicate no line breaking opportunity. The Unicode Technical Report #14 mentioned earlier has superseded the rules described by JIS X-4051.

7.3 The ‘word-break’ properties

Three properties control line breaking inside of words.

The ‘word-break-CJK’ takes the following values:

- **normal**, Keeps non-CJK scripts together (according to their own rules), while Hangul and CJK (including the Korean Hanja characters) break everywhere or according to the rules of the 'line-break' mode. Note however that using the value ‘emergency’ in the ‘word-break-wrap’ property, or the value ‘hyphenate’ in the ‘word-break-inside’ property may supercede the behavior of non-CJK scripts.
- **break-all**, Same as 'normal' for CJK and Hangul, but non-CJK scripts can break anywhere. This option is used mostly in a context where the text is predominantly using CJK characters with few non-CJK excerpts and it is desired that the text be better distributed on each line.
- **keep-all**, Same as 'normal' for all non-CJK 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.

The ‘word-break-wrap’ controls the wrapping behavior for words. It allows a word to be split arbitrarily at the end of a line if the word cannot fit in a single line. For example, this deals with the situation of very long words constrained in a fixed with container with no scrolling allowed. Possible values:

- **normal**, A word should always stay in a single line. Note however that using the value ‘break-all’ in the ‘word-break-CJK’ property, or the value ‘hyphenate’ in the ‘word-break-inside’ property can supercede this.
- **emergency**, This allows a breaking opportunity to be created even if other analysis have failed to detect another one. For example, this deals with the situation of very long words constrained in a fixed with container with no scrolling allowed.

The ‘word-break-inside’ controls the hyphenation behavior inside of words. Possible values:

- **normal**, A word should always stay in a single line. Note however that this can be superceded by using the value 'break-all' in the 'word-break-CJK' property, or the value 'emergency' in the 'word-break-wrap' property. Moreover, explicit hyphenation characters (hyphen, soft hyphen, etc...) still create line breaking opportunities.
- **hyphenate**, Words can be broken at an appropriate hyphenation point. It requires that the user agent have an hyphenation dictionary for the language of the text being broken. Setting this value activates the hyphenation engine in the user agent.

Finally, there is a ‘break-word’ shorthand property that allows combining all 3 previous properties.

8 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。

9 Text decoration

Until CSS2, the only text decorations available were available through the 'text-decoration' property exposing itself various effects underline, overline, line-through, etc... and the text-shadows property. However the text-decoration property has some limitations stemming from its syntax, which allows for multiple text-decoration formatting effects to be specified at the same time but it precludes finer control over each of those formatting effects. More specifically, it offers no way to control the color or line style of the underline, overline or line-through.

CSS3 extends the model by introducing new properties allowing additional controls over those formatting effects. CSS3 also makes turning these formatting effects on or off possible without affecting any other 'text-decoration' settings.

Furthermore, to reflect the usage of underline in East Asian vertical writing, a new control is offered on the underline positioning, this allows the underline to appear before (on the right in vertical text flow) or after (on the left in vertical text flow) the formatted text. The property is called 'text-underline-position'.

The text-decoration property itself is now a shorthand property for all these new properties.

These properties describe decorations that are added to the text of an element. If they are specified for a block-level element, it affects all inline-level descendants of the element. If they are specified for (or affects) an inline-level element, it affects all boxes generated by the element. If the element has no content or no text content (e.g., the IMG element in HTML), user agents must ignore these properties.

Finally, user agents may chose either to average thickness and positions of the 'line' text-decorations based on the children text size and baselines, or to process each children separately. The following figure shows the averaging for underline:

1st a 1st a 1st a

On these 3 segments of underline text, the underline bar is drawn lower and thicker as the ratio of large text increase for the each consecutive underlined text segment.

Note: *Typically the underline superscript text segments are averaged, while the subscript segments are not.*

Of special interest to East Asian typography, there is a new 'text-underline-style' property that can take the following values: none, solid, double, dotted, thick, dashed, dot-dash, dot-dot-dash, wave, single-accounting, double-accounting.

Here is an example of some of these styles:

Solid

Double

Dotted

Thick

Dashed

Dot-dash

Dot-dot-dash

Wave

10 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 mono-spaced 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 user agent 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).

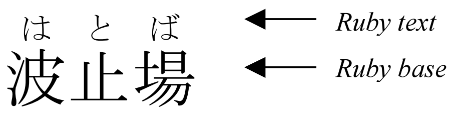
11 Ruby

11.1 Overview

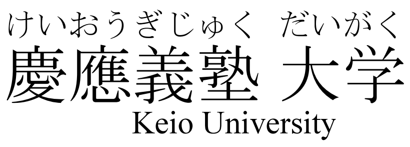
"Ruby" is the commonly used name for a run of text that appears in the immediate vicinity of another run of text, referred to as the "base", and serves as an annotation or a pronunciation guide associated with that run of text. Ruby, as used in Japanese, is described in JIS-X-4051.

The font size of ruby text is normally half the font size of the base. The name "ruby" originated from the name of the 5.5pt font size in British printing, which is about half the 10.5pt font size commonly used for normal text in East Asian text.

There are several positions where the ruby text can appear relative to its base.   It can be above, below or on the right. The following example shows the annotation above the base text (Japanese hatoba for the English word wharf)



Ruby may also appears as ‘complex’ ruby, where multiple ruby annotations can span various parts of the base text, as in the following example:

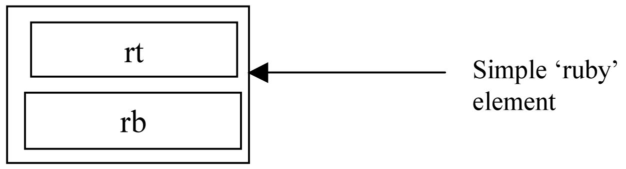


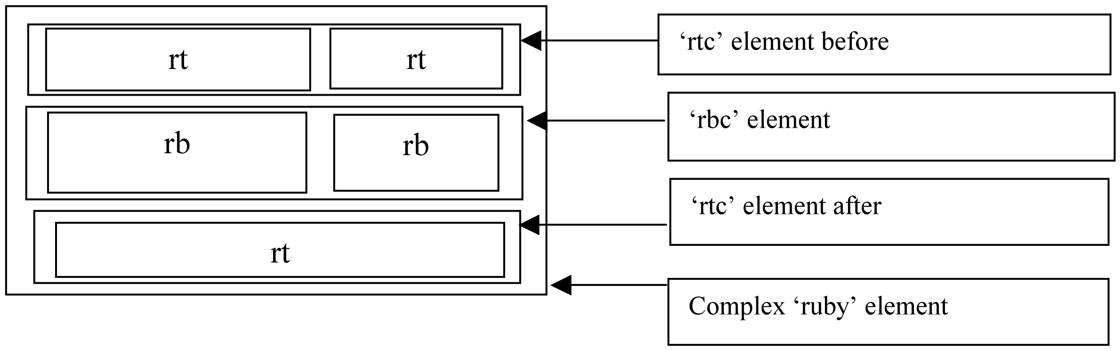
In that case the two hiragana ruby annotations (けいおうぎじゅく and だいがく) are directly on top of the two base segments, and the English annotation (Keio University) is spanning the whole base.

To describe these elements a new markup has been proposed within the context of XHTML 1.1 and XHTML Modularization. The markup takes two variants:

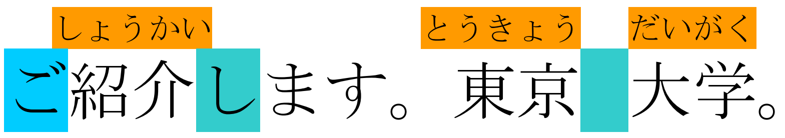
- A simple form where a ‘ruby’ element contains a ruby base ‘rb’ and a ruby text ‘rt’.
- A complex form where a ‘ruby’ element contains a ruby base container ‘rbc’ and one or two ruby text containers ‘rtc’. The ‘rbc’ element itself contains ‘rb’ elements and the ‘rtc’ elements contain ‘rt’ elements.

The following figures provide a spatial representation of the examples shown above:





Finally, it is possible for a ‘rt’ element (simple ruby) or a ‘rtc’ element (complex ruby) to hang over adjacent text. The following text shows the effect:



In the first case, the annotation hangs over the adjacent base hiragana text (shown in darker background). In the second case, the two annotations and the base text are clearly separated to avoid overhanging between the two base Kanji text sequences (the separation is slightly exaggerated for illustration purpose).

Several CSS properties have been introduced to control the rendering of these elements:

- 'ruby-position' controls the position of the ruby text elements relatively to the ruby base.
- 'ruby-align' controls the character alignment of the text contained in the elements with the narrowest advance width.
- 'ruby-overhang' controls the overhang effect described above.

These properties are described in more details in the following section.

11.2 The 'ruby-position' property

This property is used by the parents of the 'rt' elements (can either be a 'ruby' element or a 'rtc' element) to control the position of the ruby text with respect to its base. The possible values are the following:

- **before**. The ruby text appears before the base. In horizontal flow it will be above the base and in vertical ideographic mode (writing-mode: tb-rl) it will be on the right of the base. This is the most common value.
- **after**. The ruby text appears after the base (below in horizontal flow, on the left in vertical ideographic flow).
- **right**. The ruby text appears on the right of the base; this is not relative to the text flow direction (writing-mode).
- **inline**. The ruby text appears inline following the base. If some quotation marks or parenthesis are not added, this value may introduce a different meaning to the text.

11.3 The 'ruby-align' property

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. For simple ruby, the alignment is applied to the ruby child element whose content is shorter: either the 'rb' element or the 'rt' element. For complex ruby, the alignment is applied to the ruby child element(s) whose content is(are) shorter in a 'column': it can be either a 'rb' element and/or one or two 'rt' elements for each related ruby text and ruby base element within the 'rtc' and 'rbc' element. The possible values are the following:

- **auto**. The user agent determines how the ruby contents are aligned. The recommended behavior is to align wide textual content (ideographic and related characters and symbols) according to the 'distribute-space' value described below, and other textual content according to the 'center' value also described below.
- **start** and **left**. The ruby text content is aligned with the start edge of the base. This correspond to the 'left' side in horizontal layout (writing-mode:tb-lr)
- **center**. The ruby text content is 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
- **end** and **right** - The ruby text contents are aligned with the end edge of 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'.

11.4 The ruby-overhang property

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. Also the UA is free to assume a maximum amount by which ruby text may overhang adjacent text. The UA may use the JIS recommendation of using one ruby text character length as the maximum overhang length. Possible values:

- **auto**. The ruby text can overhang text adjacent to the base on either side. JIS specifies the categories of characters that ruby text can overhang. The UA is free to follow the JIS recommendation or specify its own classes of characters to overhang. This is the initial value.
- **start**. The ruby text can overhang text that precedes it. That means, for example, that ruby can overhang text that is to the left of it in horizontal LTR layout, or it can overhang text that is above it in vertical-ideographic layout.
- **end**. The ruby text can overhang text that follows it. That means, for example, that ruby can overhang text that is to the right of it in horizontal LTR layout, or it can overhang text that is below it in vertical-ideographic layout.
- **none**. The ruby text cannot overhang any text adjacent to its base, only its own base.

12 Conclusion

Many of the features exposed here will allow improving dramatically the typographical quality of worldwide text. Many of these features would not have been possible without the Unicode Standard existence. The Standard by providing guidance on character properties is making possible to implement complex text behavior like text justification, line breaking, etc. But much is still to be done for scripts and writing systems that have been recently incorporated into the Standard like Mongolian, Yi, etc… It is expected that CSS will add many new features as the understanding of international typography grows.

13 References

JIS – Line composition rules for Japanese documents JIS X 4051 1995, Japanese Standards Association (in Japanese)

CSS2 – Cascading Style Sheets, level 2 (CSS2) Specification, W3C Recommendation, Bert Bos, Håkon Wium Lie, Chris Lilley and Ian Jacobs, 12 May 1998
Available at http://www.w3.org/TR/REC-CSS2/

CSS3 – Cascading Style Sheet, level 3 (CSS3) Specification, Working Drafts, please consult the W3C main page for public documents announcements at http://www.w3.org/ or if you are a W3C member you can check the Style working group page at http://www.w3.org/Style/Group/ for further details (member id and password may be required). Further details and pointers may be available during the conference.

Ruby – Ruby Annotation, Working Draft available at http://www.w3.org/TR/1999/WD-ruby-19991217/. A newer version should be publicly available by the conference time.

XSL – Extensible Style sheet Language (XSL) Candidate Recommendation, many authors and contributors, available at http://www.w3.org/TR/xsl/index.html.