W3C Candidate Recommendation

CSS Fonts Module Level 3

W3C Candidate Recommendation 3 October 2013

This version:

http://www.w3.org/TR/2013/CR-css-fonts-3-20131003/

Latest version:

http://www.w3.org/TR/css-fonts-3/

Latest editor's draft:

http://dev.w3.org/csswg/css-fonts/ (change log)

Previous version:

http://www.w3.org/TR/2013/WD-css-fonts-3-20130711/

Issues List:

CSS3 Fonts issues in Tracker

CSS3 Fonts issues in Bugzilla

Discussion:

www-style@w3.org with subject line "`[css-fonts] … message topic …`" (archives)

Test Suite:

http://test.csswg.org/suites/css3-fonts/nightly-unstable/

Editor:

John Daggett (Mozilla)

Abstract

This CSS3 module describes how font properties are specified and how font resources are loaded dynamically. The contents of this specification are a consolidation of content previously divided into CSS3 Fonts and CSS3 Web Fonts modules. The description of font load events was moved into the CSS3 Font Load Events module.

Status of this document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.

This document was produced by the CSS Working Group as a Candidate Recommendation.

A Candidate Recommendation is a document that has been widely reviewed and is ready for implementation. W3C encourages everybody to implement this specification and return comments to the (archived) public mailing list www-style@w3.org (see instructions). When sending e-mail, please put the text "css3-fonts" in the subject, preferably like this: "[css3-fonts] *…summary of comment…*"

Publication as a Candidate Recommendation does not imply endorsement by the W3C Membership. This is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to cite this document as other than work in progress.

This document was produced by a group operating under the 5 February 2004 W3C Patent Policy. W3C maintains a public list of any patent disclosures made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) must disclose

the information in accordance with section 6 of the W3C Patent Policy.

See the section CR Exit Criteria for details on advancing this specification to W3C Recommendation. The specification will remain Candidate Recommendation at least until 1 April 2014. A test suite and implementation report are under development.

Features at risk

The following features are at risk; if interoperable implementations are not found, they may be removed to advance the other features in this specification to Proposed Recommendation:

- fallback handling for text decoration in superscript/subscript variant glyphs ('font-variant-position')
- 'font-language-override' property

Table of Contents

1 Introduction

A font provides a resource containing the visual representation of characters. At the simplest level it contains information that maps character codes to shapes (called glyphs) that represent these characters. Fonts sharing a common design style are commonly grouped into font families classified by a set of standard font properties. Within a family, the shape displayed for a given character can vary by stroke weight, slant or relative width, among others. An individual font face is described by a unique combination of these properties. For a given range of text, CSS font properties are used to select a font family and a specific font face within that family to be used when rendering that text. As a simple example, to use the bold form of Helvetica one could use:

```
body {
    font-family: Helvetica;
    font-weight: bold;
}
```

Font resources may be installed locally on the system on which a user agent is running or downloadable. For local font resources descriptive information can be obtained directly from the font resource. For downloadable font resources (sometimes referred to as web fonts), the descriptive information is included with the reference to the font resource.

Families of fonts typically don't contain a single face for each possible variation of font properties. The CSS font selection

mechanism describes how to match a given set of CSS font properties to a single font face.

2 Typography Background

This section is non-normative.

Typographic traditions vary across the globe, so there is no unique way to classify all fonts across languages and cultures. For even common Latin letters, wide variations are possible:

Figure 1. *One character, many glyph variations*

Differences in the anatomy of letterforms is one way to distinguish fonts. For Latin fonts, flourishes at the ends of a character's main strokes, or serifs, can distinguish a font from those without. Similar comparisons exist in non-Latin fonts between fonts with tapered strokes and those using primarily uniform strokes:

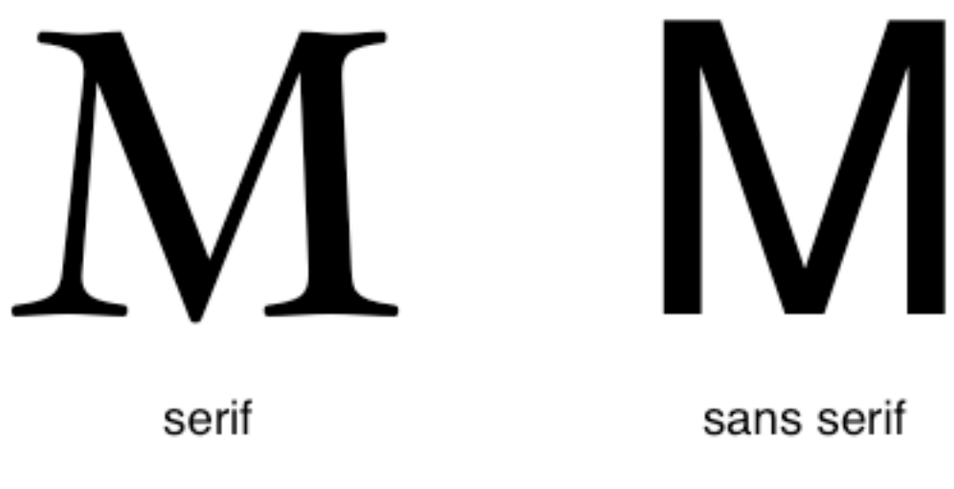


Figure 2. *Letterforms with and without serifs*

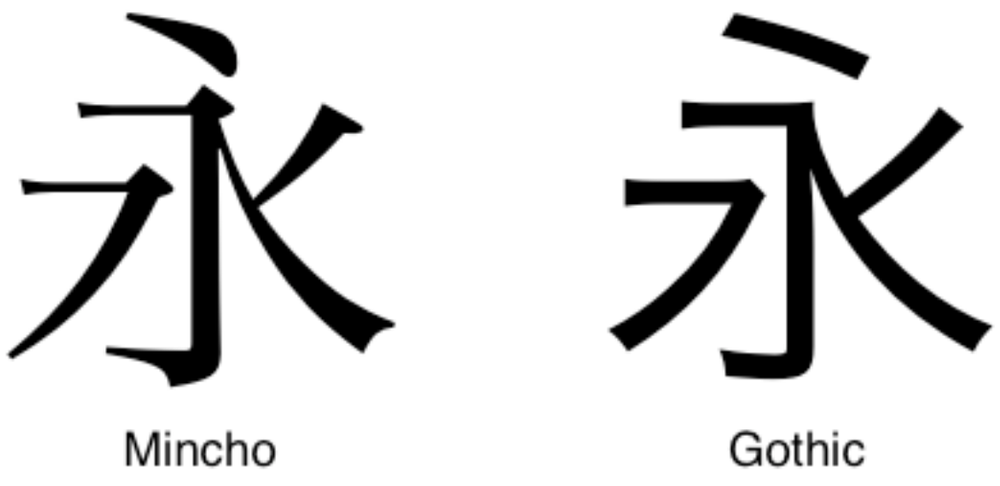


Figure 3. *Similar groupings for Japanese typefaces*

Fonts contain letterforms and the data needed to map characters to these letterforms. Often this may be a simple one-to-one mapping, but more complex mappings are also possible. The use of combining diacritic marks creates many variations for an underlying letterform:

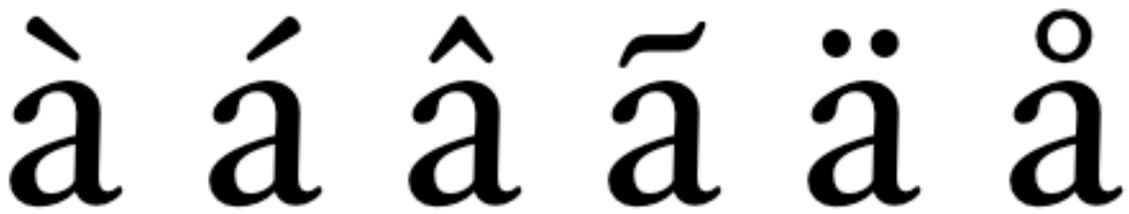


Figure 4. *Variations with diacritic marks*

A sequence of characters can be represented by a single glyph known as a ligature:

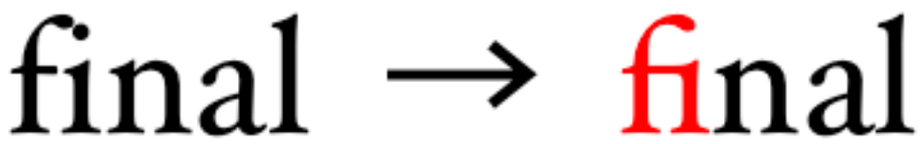


Figure 5. *Ligature example*

Visual transformations based on textual context are often stylistic option in European languages. They are required to correctly render languages like Arabic, the lam and alef characters below *must* be combined when they exist in sequence:

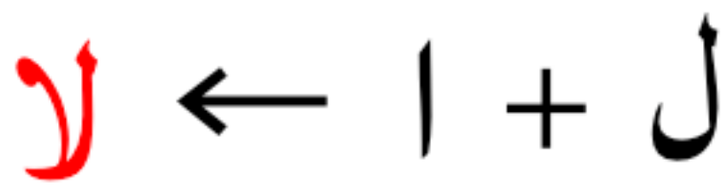


Figure 6. *Required Arabic ligature*

The relative complexity of these shaping transformations requires additional data within the font.

Sets of font faces with various stylistic variations are often grouped together into font families. In the simplest case a regular face is supplemented with bold and italic faces, but much more extensive groupings are possible. Variations in the thickness of letterform strokes, the ***weight***, and the overall proportions of the letterform, the ***width***, are most common. In the example below, each letter uses a different font face within the Univers font family. The width used increases from top to bottom and the weight increases from left to right:

Figure 7. *Weight and width variations within a single font family*

Creating fonts that support multiple scripts is a difficult task; designers need to understand the cultural traditions surrounding the use of type in different scripts and come up with letterforms that somehow share a common theme. Many languages often share a common script and each of these languages may have noticeable stylistic differences. For example, the Arabic script, when used for Persian and Urdu, exhibits significant and systematic differences in letterforms, as does Cyrillic when used with languages such as Serbian and Russian.

The *character map* of a font defines the mapping of characters to glyphs for that font. If a document contains characters not supported by the *character maps* of the fonts contained in a font family list, a user agent may use a *system font fallback* procedure to locate an appropriate font that does. If no appropriate font can be found, some form of "missing glyph" character will be rendered by the user agent. System fallback can occur when the specified list of font families does not include a font that supports a given character.

Although the *character map* of a font maps a given character to a glyph for that character, modern font technologies such as

OpenType and AAT (Apple Advanced Typography) provide ways of mapping a character to different glyphs based upon feature settings. Fonts in these formats allow these features to be embedded in the font itself and controlled by applications. Common typographic features which can be specified this way include ligatures, swashes, contextual alternates, proportional and tabular figures, and automatic fractions, to list just a few. For a visual overview of OpenType features, see the [OPENTYPE-FONT-GUIDE].

3 Basic Font Properties

The particular font face used to render a character is determined by the font family and other font properties that apply to a given element. This structure allows settings to be varied independent of each other.

3.1 Font family: the font-family property

Name:	***font-family***
Value:	[*<family-name>* | *<generic-family>*] #
Initial:	depends on user agent
Applies to:	all elements
Inherited:	yes
Percentages:	N/A
Media:	visual
Computed value:	as specified
Animatable:	no

This property specifies a prioritized list of font family names or generic family names. A font family defines a set of faces that vary in weight, width or slope. CSS uses the combination of a family name with other style attributes to select an individual face. Using this selection mechanism, rather than selecting a face via the style name as is often done in design applications, allows some degree of regularity in textual display when fallback occurs.

> Designers should note that the CSS definition of font attributes used for selection are explicitly not intended to define a font taxonomy. A type designer's idea of a family may often extend to a set of faces that vary along axes other than just the standard axes of weight, width and slope. A family may extend to include both a set of serif faces and a set of sans-serif faces or vary along axes that are unique to that family. The CSS font selection mechanism merely provides a way to determine the “closest” substitute when substitution is necessary.

Unlike other CSS properties, component values are a comma-separated list indicating alternatives. A user agent iterates through the list of family names until it matches an available font that contains a glyph for the character to be rendered. This allows for differences in available fonts across platforms and for differences in the range of characters supported by individual fonts.

A font family name only specifies a name given to a set of font faces, it does not specify an individual face. For example, given the availability of the fonts below, Futura would match but Futura Medium would not:

Figure 8. *Family and individual face names*

Consider the example below:

> EXAMPLE 1
>
> ```
> body {
>     font-family: Helvetica, Verdana, sans-serif;
> }
> ```
>
> If Helvetica is available it will be used when rendering. If neither Helvetica nor Verdana is present, then the user-agent-defined sans serif font will be used.

There are two types of font family names:

<family-name>
: The name of a font family of choice such as Helvetica or Verdana in the previous example.

<generic-family>
: The following generic family keywords are defined: ‘serif’, ‘sans-serif’, ‘cursive’, ‘fantasy’, and ‘monospace’. These keywords can be used as a general fallback mechanism when an author's desired font choices are not available. As keywords, they must not be quoted. Authors are encouraged to append a generic font family as a last alternative for improved robustness.

Font family names other than generic families must either be given quoted as strings, or unquoted as a sequence of one or more identifiers. This means most punctuation characters and digits at the start of each token must be escaped in unquoted font family names.

To illustrate this, the following declarations are invalid:

```
font-family: Red/Black, sans-serif;
font-family: "Lucida" Grande, sans-serif;
font-family: Ahem!, sans-serif;
font-family: test@foo, sans-serif;
font-family: #POUND, sans-serif;
font-family: Hawaii 5-0, sans-serif;
```

If a sequence of identifiers is given as a font family name, the computed value is the name converted to a string by joining all the identifiers in the sequence by single spaces.

To avoid mistakes in escaping, it is recommended to quote font family names that contain white space, digits, or punctuation characters other than hyphens:

```
body { font-family: "New Century Schoolbook", serif }

<BODY STYLE="font-family: '21st Century', fantasy">
```

Font family *names* that happen to be the same as a keyword value (‘inherit’, ‘serif’, ‘sans-serif’, ‘monospace’, ‘fantasy’, and ‘cursive’) must be quoted to prevent confusion with the keywords with the same names. The keywords ‘initial’ and ‘default’ are reserved for future use and must also be quoted when used as font names. UAs must not consider these keywords as matching the *<family-name>* type.

The precise way a set of fonts are grouped into font families varies depending upon the platform font management API's. The Windows GDI API only allows four faces to be grouped into a family while the DirectWrite API and API's on OSX and other platforms support font families with a variety of weights, widths and slopes (see Appendix A for more details).

Some font formats allow fonts to carry multiple localizations of the family name. User agents must recognize and correctly match all of these names independent of the underlying platform localization, system API used or document encoding:

GulimChe	굴림체
Hiragino Kaku Gothic Pro	ヒラギノ角ゴ Pro
Meiryo	メイリオ
MingLiU	細明體
MS Mincho	ＭＳ 明朝
Raanana	רעננה

Figure 9. *Localized family names*

The details of localized font family name matching and the corresponding issues of case sensitivity are described below in the font matching section.

3.1.1 Generic font families

All five generic font families are defined to exist in all CSS implementations (they need not necessarily map to five distinct actual fonts). User agents should provide reasonable default choices for the generic font families, which express the characteristics of each family as well as possible within the limits allowed by the underlying technology. User agents are encouraged to allow users to select alternative choices for the generic fonts.

serif

Serif fonts represent the formal text style for a script. This often means but is not limited to glyphs that have finishing strokes, flared or tapering ends, or have actual serifed endings (including slab serifs). Serif fonts are typically proportionately-spaced. They often display a greater variation between thick and thin strokes than fonts from the ‘sans-serif’ generic font family. CSS uses the term ‘serif’ to apply to a font for any script, although other names may be more familiar for particular scripts, such as Mincho (Japanese), Sung, Song or Kai (Chinese), Batang (Korean). For Arabic, the Naskh style would correspond to ‘serif’ more due to its typographic role rather than its actual design style. Any font that is so described may be used to represent the generic ‘serif’ family.

Figure 10. *Sample serif fonts*

sans-serif

Glyphs in sans-serif fonts, as the term is used in CSS, are generally low contrast (vertical and horizontal stems have the close to the same thickness) and have stroke endings that are plain -- without any flaring, cross stroke, or other ornamentation. Sans-serif fonts are typically proportionately-spaced. They often have little variation between thick and thin strokes, compared to fonts from the ‘serif’ family. CSS uses the term ‘sans-serif’ to apply to a font for any script, although other names may be more familiar for particular scripts, such as Gothic (Japanese), Hei (Chinese), or Gulim (Korean). Any font that is so described may be used to represent the generic ‘sans-serif’ family.

Figure 11. *Sample sans-serif fonts*

cursive

Glyphs in cursive fonts generally use a more informal script style, and the result looks more like handwritten pen or brush writing than printed letterwork. CSS uses the term ‘cursive’ to apply to a font for any script, although other names such as Chancery, Brush, Swing and Script are also used in font names.

Figure 12. *Sample cursive fonts*

fantasy

Fantasy fonts are primarily decorative or expressive fonts that contain decorative or expressive representations of characters. These do not include Pi or Picture fonts which do not represent actual characters.

Figure 13. *Sample fantasy fonts*

monospace

The sole criterion of a monospace font is that all glyphs have the same fixed width. This is often used to render samples of computer code.

Figure 14. *Sample monospace fonts*

3.2 Font weight: the font-weight property

Name:	***font-weight***
Value:	normal | bold | bolder | lighter | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900
Initial:	normal
Applies to:	all elements
Inherited:	yes
Percentages:	N/A
Media:	visual
Computed value:	numeric weight value (see description)
Animatable:	as font weight

The ‘font-weight’ property specifies the weight of glyphs in the font, their degree of blackness or stroke thickness.

Values have the following meanings:

100 to 900
: These values form an ordered sequence, where each number indicates a weight that is at least as dark as its predecessor. These roughly correspond to the commonly used weight names below:

- 100 - Thin
- 200 - Extra Light (Ultra Light)
- 300 - Light
- 400 - Normal
- 500 - Medium
- 600 - Semi Bold (Demi Bold)
- 700 - Bold
- 800 - Extra Bold (Ultra Bold)
- 900 - Black (Heavy)

normal
: Same as ‘400’.

bold
: Same as ‘700’.

bolder
: Specifies a bolder weight than the inherited value.

lighter
: Specifies a lighter weight than the inherited value.

Font formats that use a scale other than a nine-step scale should map their scale onto the CSS scale so that 400 roughly corresponds with a face that would be labeled as Regular, Book, Roman and 700 roughly matches a face that would be labeled as Bold. Or weights may be inferred from the style names, ones that correspond roughly with the scale above. The scale is relative, so a face with a larger weight value must never appear lighter. If style names are used to infer weights, care should be taken to handle variations in style names across locales.

Quite often there are only a few weights available for a particular font family. When a weight is specified for which no face exists, a face with a nearby weight is used. In general, bold weights map to faces with heavier weights and light weights map to faces with lighter weights (see the font matching section below for a precise definition). The examples here illustrate which face is used for different weights, grey indicates a face for that weight does not exist so a face with a nearby weight is used:

Figure 15. *Weight mappings for a font family with 400, 700 and 900 weight faces*

Figure 16. *Weight mappings for a font family with 300 and 600 weight faces*

Although the practice is not well-loved by typographers, bold faces are often synthesized by user agents for faces that lack actual bold faces. For the purposes of style matching, these faces must be treated as if they exist within the family. Authors can explicitly avoid this behavior by using the ‘font-synthesis’ property.

Specified values of ‘bolder’ and ‘lighter’ indicate weights relative to the weight of the parent element. The computed weight is calculated based on the inherited ‘font-weight’ value using the chart below.

Inherited value	bolder	lighter
100	400	100
200	400	100
300	400	100
400	700	100
500	700	100
600	900	400
700	900	400
800	900	700
900	900	700

The table above is equivalent to selecting the next relative bolder or lighter face, given a font family containing normal and bold faces along with a thin and a heavy face. Authors who desire finer control over the exact weight values used for a given element may use numerical values instead of relative weights.

3.3 Font width: the font-stretch property

Name:	***font-stretch***
Value:	normal | ultra-condensed | extra-condensed | condensed | semi-condensed | semi-expanded | expanded | extra-expanded | ultra-expanded
Initial:	normal
Applies to:	all elements
Inherited:	yes
Percentages:	N/A
Media:	visual
Computed value:	as specified
Animatable:	as font stretch

The ‘font-stretch’ property selects a normal, condensed, or expanded face from a font family. Absolute keyword values have the following ordering, from narrowest to widest:

- ***ultra-condensed***
- ***extra-condensed***
- ***condensed***
- ***semi-condensed***
- ***normal***
- ***semi-expanded***
- ***expanded***
- ***extra-expanded***
- ***ultra-expanded***

When a face does not exist for a given width, normal or condensed values map to a narrower face, otherwise a wider face. Conversely, expanded values map to a wider face, otherwise a narrower face. The figure below shows how the nine font-stretch property settings affect font selection for font family containing a variety of widths, grey indicates a width for which no face exists and a different width is substituted:

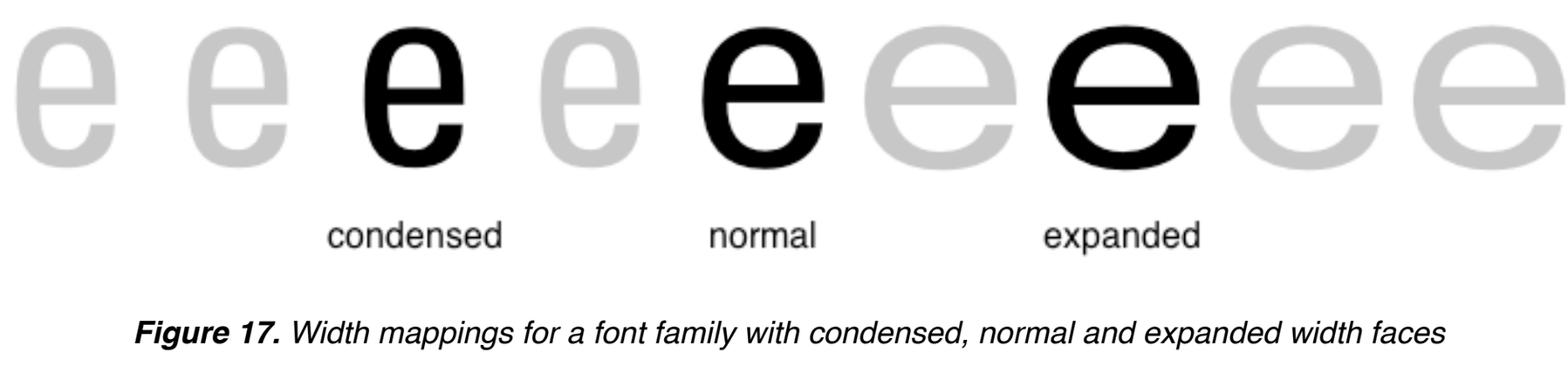


Figure 17. *Width mappings for a font family with condensed, normal and expanded width faces*

Animation of font stretch: Font stretch is interpolated in discrete steps. The interpolation happens as though the ordered values are equally spaced real numbers. The interpolation result is rounded to the nearest value, with values exactly halfway between two values rounded towards the later value in the list above.

3.4 Font style: the font-style property

Name:	***font-style***
Value:	normal | italic | oblique
Initial:	normal
Applies to:	all elements
Inherited:	yes
Percentages:	N/A
Media:	visual
Computed value:	as specified
Animatable:	no

The ‘font-style’ property allows italic or oblique faces to be selected. Italic forms are generally cursive in nature while oblique faces are typically sloped versions of the regular face. Oblique faces can be simulated by artificially sloping the glyphs of the regular face. Compare the artificially sloped renderings of Palatino ‘a’ and Baskerville ‘N’ in grey with the actual italic versions:

Figure 18. *Artificial sloping versus real italics*

Values have the following meanings:

normal
: selects a face that is classified as a normal face, one that is neither italic or obliqued

italic
: selects a font that is labeled as an italic face, or an oblique face if one is not

oblique
: selects a font that is labeled as an oblique face, or an italic face if one is not

If no italic or oblique face is available, oblique faces can be synthesized by rendering non-obliqued faces with an artificial obliquing operation. The use of these artificially obliqued faces can be disabled using the ‘font-synthesis’ property. The details of the obliquing operation are not explicitly defined.

> Authors should also be aware that synthesized approaches may not be suitable for scripts like Cyrillic, where italic forms are very different in shape. It is always better to use an actual italic font rather than rely on a synthetic version.

Many scripts lack the tradition of mixing a cursive form within text rendered with a normal face. Chinese, Japanese and Korean fonts almost always lack italic or oblique faces. Fonts that support a mixture of scripts will sometimes omit specific scripts such as Arabic from the set of glyphs supported in the italic face. User agents should be careful about making *character map* assumptions across faces when implementing support for *system font fallback*.

3.5 Font size: the font-size property

Name:	***font-size***
Value:	*<absolute-size>* | *<relative-size>* | *<length>* | *<percentage>*
Initial:	medium
Applies to:	all elements
Inherited:	yes
Percentages:	refer to parent element's font size
Media:	visual
Computed value:	absolute length
Animatable:	as length

This property indicates the desired height of glyphs from the font. For scalable fonts, the font-size is a scale factor applied to the EM unit of the font. (Note that certain glyphs may bleed outside their EM box.) For non-scalable fonts, the font-size is converted into absolute units and matched against the declared font-size of the font, using the same absolute coordinate space for both of the matched values. Values have the following meanings:

<absolute-size>

An *<absolute-size>* keyword refers to an entry in a table of font sizes computed and kept by the user agent. Possible values are:

[xx-small | x-small | small | medium | large | x-large | xx-large]

<relative-size>

A *<relative-size>* keyword is interpreted relative to the table of font sizes and the computed 'font-size' of the parent element. Possible values are:

[larger | smaller]

For example, if the parent element has a font size of 'medium', a value of 'larger' will make the font size of the current element be 'large'. If the parent element's size is not close to a table entry, the user agent is free to interpolate between table entries or round off to the closest one. The user agent may have to extrapolate table values if the numerical value goes beyond the keywords.

<length>

A length value specifies an absolute font size (independent of the user agent's font table). Negative lengths are illegal.

<percentage>

A percentage value specifies an absolute font size relative to the parent element's font size. Use of percentage values, or values in 'em's, leads to more robust and cascadable style sheets.

The following table provides user agent guidelines for the absolute-size scaling factor and their mapping to HTML heading and absolute font-sizes. The 'medium' value is used as the reference middle value. The user agent may fine-tune these values for different fonts or different types of display devices.

CSS absolute-size values	xx-small	x-small	small	medium	large	x-large	xx-large	
scaling factor	3/5	3/4	8/9	1	6/5	3/2	2/1	3/1
HTML headings	h6		h5	h4	h3	h2	h1	
HTML font sizes	1		2	3	4	5	6	7

Note 1. *To preserve readability, an UA applying these guidelines should nevertheless avoid creating font-size resulting in less than 9 device pixels per EM unit on a computer display.*

> ***Note 2.*** *In CSS1, the suggested scaling factor between adjacent indexes was 1.5 which user experience proved to be too large. In CSS2, the suggested scaling factor for computer screen between adjacent indexes was 1.2 which still created issues for the small sizes. The new scaling factor varies between each index to provide a better readability.*

The actual value of this property may differ from the computed value due a numerical value on 'font-size-adjust' and the unavailability of certain font sizes.

Child elements inherit the computed 'font-size' value (otherwise, the effect of 'font-size-adjust' would compound).

EXAMPLE 2

```
p { font-size: 12pt; }
blockquote { font-size: larger }
em { font-size: 150% }
em { font-size: 1.5em }
```

3.6 Relative sizing: the font-size-adjust property

Name:	***font-size-adjust***
Value:	none | *<number>*
Initial:	none
Applies to:	all elements
Inherited:	yes
Percentages:	N/A
Media:	visual
Computed value:	as specified
Animatable:	as number

For any given font size, the apparent size and legibility of text varies across fonts. For scripts such as Latin or Cyrillic that distinguish between upper and lowercase letters, the relative height of lowercase letters compared to their uppercase counterparts is a determining factor of legibility. This is commonly referred to as the ***aspect value***. Precisely defined, it is equal to the x-height of a font divided by the font size.

In situations where font fallback occurs, fallback fonts may not share the same aspect value as the desired font family and will thus appear less readable. The 'font-size-adjust' property is a way to preserve the readability of text when font fallback occurs. It does this by adjusting the font-size so that the x-height is the same regardless of the font used.

EXAMPLE 3

The style defined below defines Verdana as the desired font family, but if Verdana is not available Futura or Times will be used.

```
p {
    font-family: Verdana, Futura, Times;
}

<p>Lorem ipsum dolor sit amet, ...</p>
```

Verdana has a relatively high aspect value, lowercase letters are relatively tall compared to uppercase letters, so at small sizes text appears legible. Times has a lower aspect value and so if fallback occurs, the text will be less legible at small sizes than Verdana.

How text rendered in each of these fonts compares is shown below, the columns show text rendered in Verdana, Futura and Times. The same font-size value is used across cells within each row and red lines are included to show the differences in x-height. In the upper half each row is rendered in the same font-size value. The same is true for the lower half but in this half the 'font-size-adjust' property is also set so that the actual font size is adjusted to preserve the x-height across each row. Note how small text remains relatively legible across each row in the lower half.

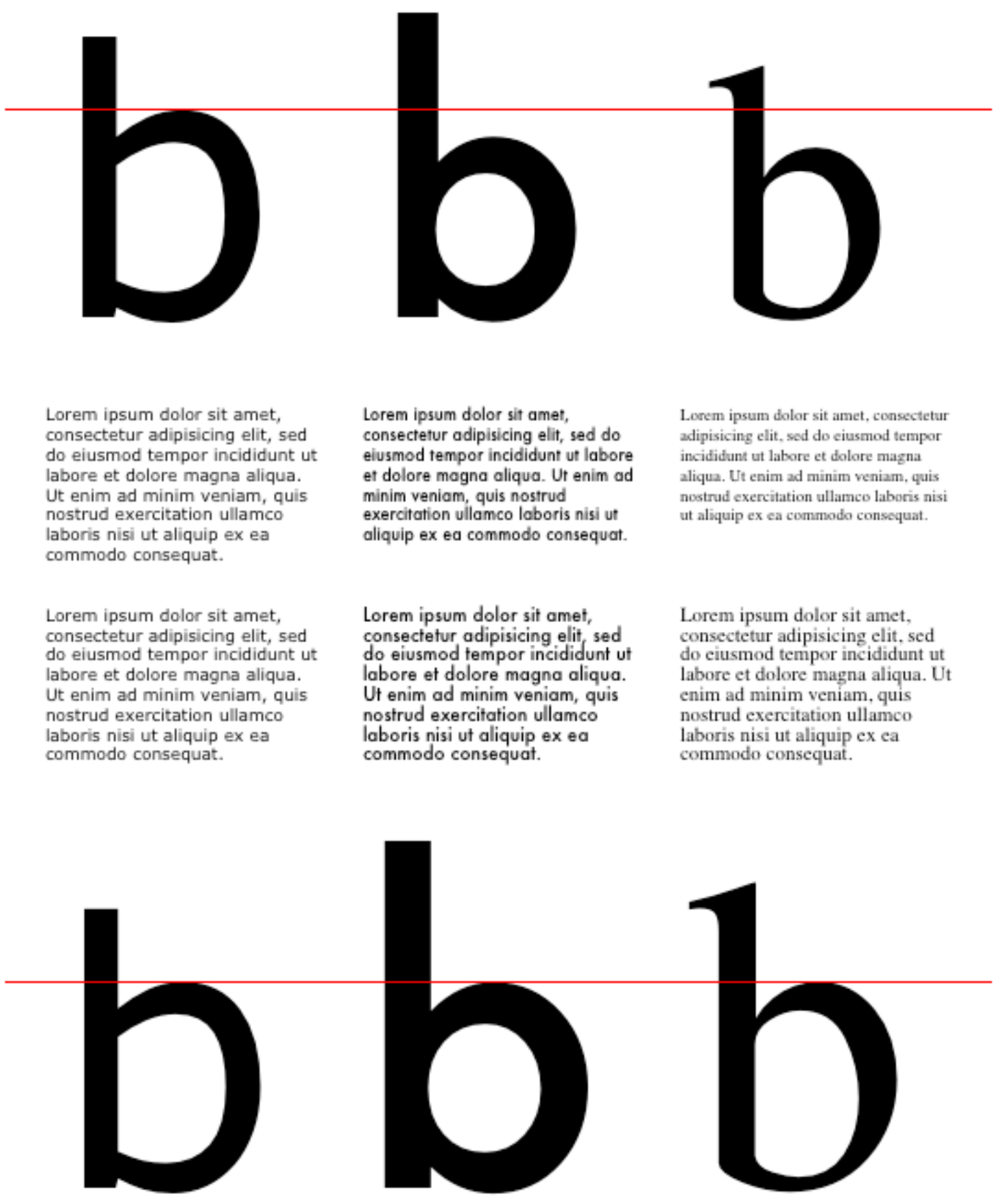


Figure 19. *Text with and without the use of 'font-size-adjust'*

This property allows authors to specify an aspect value for an element that will effectively preserve the x-height of the first choice font, whether it is substituted or not. Values have the following meanings:

none

Do not preserve the font's x-height.

<number>

Specifies the aspect value used in the calculation below to calculate the adjusted font size:

```
c  =  ( a / a' ) s
```

where:

```
s  =  font-size value
a  =  aspect value as specified by the 'font-size-adjust' property
a' =  aspect value of actual font
c  =  adjusted font-size to use
```

This value applies to any font that is selected but in typical usage it should be based on the aspect value of the first font in the font-family list. If this is specified accurately, the `(a/a')` term in the formula above is effectively 1 for the first font and no adjustment occurs. If the value is specified inaccurately, text rendered using the first font in the family list will display differently in older user agents that don't support ‘font-size-adjust’.

The value of ‘font-size-adjust’ affects the used value of ‘font-size’ but does not affect the computed value. It affects the size of relative units that are based on font metrics of the first available font such as `ex` and `ch` but does not affect the size of `em` units. Since numeric values of ‘line-height’ refer to the computed size of ‘font-size’, ‘font-size-adjust’ does not affect the used value of ‘line-height’.

> In CSS, authors often specify ‘line-height’ as a multiple of the ‘font-size’. Since the ‘font-size-adjust’ property affects the used value of ‘font-size’, authors should take care setting the line height when ‘font-size-adjust’ is used. Setting the line height too tightly can result in overlapping lines of text in this situation.

Authors can calculate the aspect value for a given font by comparing spans with the same content but different ‘font-size-adjust’ properties. If the same font-size is used, the spans will match when the ‘font-size-adjust’ value is accurate for the given font.

EXAMPLE 4

Two spans with borders are used to determine the aspect value of a font. The 'font-size' is the same for both spans but the 'font-size-adjust' property is specified only for the right span. Starting with a value of 0.5, the aspect value can be adjusted until the borders around the two letters line up.

```
p {
    font-family: Futura;
    font-size: 500px;
}

span {
    border: solid 1px red;
}

.adjust {
    font-size-adjust: 0.5;
}

<p><span>b</span><span class="adjust">b</span></p>
```

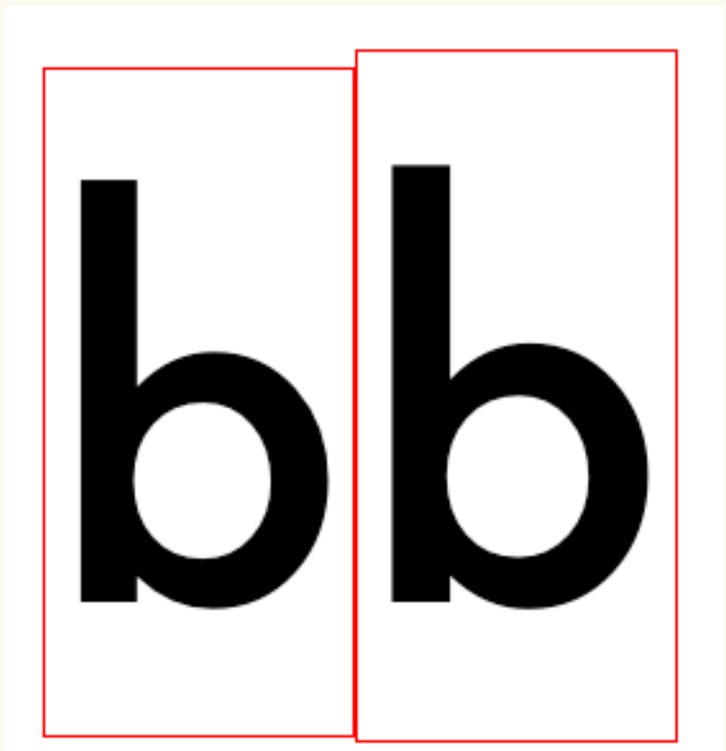


Figure 20. *Futura with an* aspect value *of 0.5*

The box on the right is a bit bigger than the one on the left, so the aspect value of this font is something less than 0.5. Adjust the value until the boxes align.

3.7 Shorthand font property: the font property

Name:	***font***
Value:	[[<*'font-style'*> || <*font-variant-css21*> || <*'font-weight'*> || <*'font-stretch'*>]? <*'font-size'*> [/ <*'line-height'*>]? <*'font-family'*>] | caption | icon | menu | message-box | small-caption | status-bar
Initial:	see individual properties
Applies to:	all elements
Inherited:	yes
Percentages:	see individual properties
Media:	visual
Computed value:	see individual properties
Animatable:	see individual properties

The 'font' property is, except as described below, a shorthand property for setting 'font-style', 'font-variant', 'font-weight', 'font-stretch', 'font-size', 'line-height', 'font-family' at the same place in the stylesheet. Values for the 'font-variant' property may also be included but only those supported in CSS 2.1, none of the 'font-variant' values added in this specification can be used in the 'font' shorthand:

<font-variant-css21> = [normal | small-caps]

The syntax of this property is based on a traditional typographical shorthand notation to set multiple properties related to fonts.

All subproperties of the 'font' property are first reset to their initial values, including those listed above plus 'font-size-adjust', 'font-kerning', subproperties of 'font-variant' and and 'font-language-override'. Then, those properties that are given explicit values in the 'font' shorthand are set to those values. For a definition of allowed and initial values, see the previously defined properties. For reasons of backwards compatibility, it is not possible to set 'font-size-adjust' to anything other than its initial value using the 'font' shorthand property; instead, use the individual property.

EXAMPLE 5

```
p { font: 12pt/14pt sans-serif }
p { font: 80% sans-serif }
p { font: x-large/110% "new century schoolbook", serif }
p { font: bold italic large Palatino, serif }
p { font: normal small-caps 120%/120% fantasy }
p { font: condensed oblique 12pt "Helvetica Neue", serif; }
```

In the second rule, the font size percentage value ('80%') refers to the computed 'font-size' of the parent element. In the third rule, the line height percentage ('110%') refers to the font size of the element itself.

The first three rules do not specify the 'font-variant' and 'font-weight' explicitly, so these properties receive their initial values ('normal'). Notice that the font family name "new century schoolbook", which contains spaces, is enclosed in quotes. The fourth rule sets the 'font-weight' to 'bold', the 'font-style' to 'italic', and implicitly sets 'font-variant' to 'normal'.

The fifth rule sets the 'font-variant' ('small-caps'), the 'font-size' (120% of the parent's font size), the 'line-height' (120% of the font size) and the 'font-family' ('fantasy'). It follows that the keyword 'normal' applies to the two remaining properties: 'font-style' and 'font-weight'.

The sixth rule sets the 'font-style', 'font-stretch', 'font-size', and 'font-family', the other font properties being set to their initial values.

Since the 'font-stretch' property was not defined in CSS 2.1, when using 'font-stretch' values within 'font' rules, authors should include a extra version compatible with older user agents:

```
p {
  font: 80% sans-serif;   /* for older user agents */
  font: condensed 80% sans-serif;
}
```

The following values refer to system fonts:

caption
: The font used for captioned controls (e.g., buttons, drop-downs, etc.).

icon
: The font used to label icons.

menu
: The font used in menus (e.g., dropdown menus and menu lists).

message-box
: The font used in dialog boxes.

small-caption
: The font used for labeling small controls.

status-bar
: The font used in window status bars.

System fonts may only be set as a whole; that is, the font family, size, weight, style, etc. are all set at the same time. These values may then be altered individually if desired. If no font with the indicated characteristics exists on a given platform, the user agent should either intelligently substitute (e.g., a smaller version of the ‘caption’ font might be used for the ‘small-caption’ font), or substitute a user agent default font. As for regular fonts, if, for a system font, any of the individual properties are not part of the operating system's available user preferences, those properties should be set to their initial values.

That is why this property is "almost" a shorthand property: system fonts can only be specified with this property, not with ‘font-family’ itself, so ‘font’ allows authors to do more than the sum of its subproperties. However, the individual properties such as ‘font-weight’ are still given values taken from the system font, which can be independently varied.

Note that the keywords used for the system fonts listed above are only treated as keywords when they occur in the initial position, in other positions the same string is treated as part of the font family name:

```
font: menu;         /* use the font settings for system menus */
font: large menu;   /* use a font family named "menu" */
```

EXAMPLE 6

```
button { font: 300 italic 1.3em/1.7em "FB Armada", sans-serif }
button p { font: menu }
button p em { font-weight: bolder }
```

If the font used for dropdown menus on a particular system happened to be, for example, 9-point Charcoal, with a weight of 600, then P elements that were descendants of BUTTON would be displayed as if this rule were in effect:

```
button p { font: 600 9pt Charcoal }
```

Because the ‘font’ shorthand resets to its initial value any property not explicitly given a value, this has the same effect as this declaration:

```
button p {
  font-style: normal;
  font-variant: normal;
  font-weight: 600;
  font-size: 9pt;
  line-height: normal;
  font-family: Charcoal
}
```

3.8 Controlling synthetic faces: the font-synthesis property

Name:	***font-synthesis***
Value:	none | [weight || style]
Initial:	weight style
Applies to:	all elements
Inherited:	yes
Percentages:	N/A
Media:	visual
Computed value:	as specified
Animatable:	no

This property controls whether user agents are allowed to synthesize bold or oblique font faces when a font family lacks bold or italic faces. If ‘weight’ is not specified, user agents must not synthesize bold faces and if ‘style’ is not specified user agents must not synthesize italic faces. A value of ‘none’ disallows all synthetic faces.

EXAMPLE 7

The style rule below disables the use of synthetically obliqued Arabic:

```
*:lang(ar) { font-synthesis: none; }
```

4 Font Resources

4.1 The `@font-face` rule

The `@font-face` rule allows for linking to fonts that are automatically fetched and activated when needed. This allows authors to select a font that closely matches the design goals for a given page rather than limiting the font choice to a set of fonts available on a given platform. A set of font descriptors define the location of a font resource, either locally or externally, along with the style characteristics of an individual face. Multiple `@font-face` rules can be used to construct font families with a variety of faces. Using CSS font matching rules, a user agent can selectively download only those faces that are needed for a given piece of text.

The `@font-face` rule consists of the `@font-face` at-keyword followed by a block of descriptor declarations. In terms of the grammar, this specification defines the following productions:

```
font_face_rule
  : FONT_FACE_SYM S* '{' S* descriptor_declaration? [ ';' S* descriptor_declaration? ]* '}' S*
  ;

descriptor_declaration
  : property ':' S* expr
  ;
```

The following new definitions are introduced:

```
-    -|\\0{0,4}2d(\r\n|[ \t\r\n\f])?
F    f|\\0{0,4}(46|66)(\r\n|[ \t\r\n\f])?
```

The following new token is introduced:

```
@{F}{O}{N}{T}{-}{F}{A}{C}{E} {return FONT_FACE_SYM;}
```

Each `@font-face` rule specifies a value for every font descriptor, either implicitly or explicitly. Those not given explicit values in the

rule take the initial value listed with each descriptor in this specification. These descriptors apply solely within the context of the `@font-face` rule in which they are defined, and do not apply to document language elements. There is no notion of which elements the descriptors apply to or whether the values are inherited by child elements. When a given descriptor occurs multiple times in a given `@font-face` rule, only the last descriptor declaration is used and all prior declarations for that descriptor are ignored.

EXAMPLE 8

To use a downloadable font called Gentium:

```
@font-face {
  font-family: Gentium;
  src: url(http://example.com/fonts/Gentium.woff);
}

p { font-family: Gentium, serif; }
```

The user agent will download Gentium and use it when rendering text within paragraph elements. If for some reason the site serving the font is unavailable, the default serif font will be used.

A given set of `@font-face` rules define a set of fonts available for use within the documents that contain these rules. When font matching is done, fonts defined using these rules are considered before other available fonts on a system.

Downloaded fonts are only available to documents that reference them. The process of activating these fonts must not make them available to other applications or to documents that don't directly link to the same font. User agent implementers might consider it convenient to use downloaded fonts when rendering characters in other documents for which no other available font exists as part of the *system font fallback* procedure. However, this would cause a security leak since the contents of one page would be able to affect other pages, something an attacker could use as an attack vector. These restrictions do not affect caching behavior, fonts are cached the same way other web resources are cached.

This at-rule follows the forward-compatible parsing rules of CSS. Like properties in a declaration block, declarations of any descriptors that are not supported by the user agent must be ignored. `@font-face` rules require a font-family and src descriptor; if either of these are missing, the `@font-face` rule is invalid and must be ignored entirely.

In cases where user agents have limited platform resources or implement the ability to disable downloadable font resources, `@font-face` rules must simply be ignored; the behavior of individual descriptors as defined in this specification should not be altered.

4.2 Font family: the font-family descriptor

Name:	***font-family***
Value:	*<family-name>*
Initial:	N/A

This descriptor defines the font family name that will be used in all CSS font family name matching. It is required for the `@font-face` rule to be valid. It overrides the font family names contained in the underlying font data. If the font family name is the same as a font family available in a given user's environment, it effectively hides the underlying font for documents that use the stylesheet. This permits a web author to freely choose font-family names without worrying about conflicts with font family names present in a given user's environment. Likewise, platform substitutions for a given font family name must not be used.

4.3 Font reference: the src descriptor

Name:	***src***
Value:	[<url> [format(<string> #)]? | *<font-face-name>*] #

Initial:	N/A

This descriptor specifies the resource containing font data. It is required for the `@font-face` rule to be valid. Its value is a prioritized, comma-separated list of external references or locally-installed font face names. When a font is needed the user agent iterates over the set of references listed, using the first one it can successfully activate. Fonts containing invalid data or local font faces that are not found are ignored and the user agent loads the next font in the list.

As with other URLs in CSS, the URL may be relative, in which case it is resolved relative to the location of the style sheet containing the `@font-face` rule. In the case of SVG fonts, the URL points to an element within a document containing SVG font definitions. If the element reference is omitted, a reference to the first defined font is implied. Similarly, font container formats that can contain more than one font must load one and only one of the fonts for a given `@font-face` rule. Fragment identifiers are used to indicate which font to load. If a container format lacks a defined fragment identifier scheme, implementations should use a simple 1-based indexing scheme (e.g. "font-collection#1" for the first font, "font-collection#2" for the second font).

```
src: url(fonts/simple.woff);   /* load simple.woff relative to stylesheet location */
src: url(/fonts/simple.woff);  /* load simple.woff from absolute location */
src: url(fonts.svg#simple);    /* load SVG font with id 'simple' */
```

External references consist of a URL, followed by an optional hint describing the format of the font resource referenced by that URL. The format hint contains a comma-separated list of format strings that denote well-known font formats. Conformant user agents must skip downloading a font resource if the format hints indicate only unsupported or unknown font formats. If no format hints are supplied, the user agent should download the font resource.

```
/* load WOFF font if possible, otherwise use OpenType font */
@font-face {
  font-family: bodytext;
  src: url(ideal-sans-serif.woff) format("woff"),
       url(basic-sans-serif.ttf) format("opentype");
}
```

Format strings defined by this specification:

String	Font Format	Common extensions
"woff"	WOFF (Web Open Font Format)	.woff
"truetype"	TrueType	.ttf
"opentype"	OpenType	.ttf, .otf
"embedded-opentype"	Embedded OpenType	.eot
"svg"	SVG Font	.svg, .svgz

Given the overlap in common usage between TrueType and OpenType, the format hints "truetype" and "opentype" must be considered as synonymous; a format hint of "opentype" does not imply that the font contains Postscript CFF style glyph data or that it contains OpenType layout information (see Appendix A for more background on this).

When authors would prefer to use a locally available copy of a given font and download it if it's not, `local()` can be used. The locally-installed ***<font-face-name>*** argument to `local()` is a format-specific string that uniquely identifies a single font face within a larger family. The syntax for a *<font-face-name>* is a unique font face name enclosed by "local(" and ")". The name can optionally be enclosed in quotes. If unquoted, the unquoted font family name processing conventions apply; the name must be a sequence of identifiers separated by whitespace which is converted to a string by joining the identifiers together separated by a single space.

```
/* regular face of Gentium */
@font-face {
  font-family: MyGentium;
  src: local(Gentium),    /* use locally available Gentium */
       url(Gentium.woff); /* otherwise, download it */
}
```

For OpenType and TrueType fonts, this string is used to match only the Postscript name or the full font name in the name table of

locally available fonts. Which type of name is used varies by platform and font, so authors should include both of these names to assure proper matching across platforms. Platform substitutions for a given font name must not be used.

```
/* bold face of Gentium */
@font-face {
  font-family: MyGentium;
  src: local(Gentium Bold),    /* full font name */
       local(Gentium-Bold),    /* Postscript name */
       url(GentiumBold.woff);  /* otherwise, download it */
  font-weight: bold;
}
```

Just as a `@font-face` rule specifies the characteristics of a single font within a family, the unique name used with `local()` specifies a single font, not an entire font family. Defined in terms of OpenType font data, the Postscript name is found in the font's name table, in the name record with nameID = 6 (see [OPENTYPE] for more details). The Postscript name is the commonly used key for all fonts on OSX and for Postscript CFF fonts under Windows. The full font name (nameID = 4) is used as a unique key for fonts with TrueType glyphs on Windows.

For OpenType fonts with multiple localizations of the full font name, the US English version is used (language ID = 0x409 for Windows and language ID = 0 for Macintosh) or the first localization when a US English full font name is not available (the OpenType specification recommends that all fonts minimally include US English names). User agents that also match other full font names, e.g. matching the Dutch name when the current system locale is set to Dutch, are considered non-conformant. This is done not to prefer English but to avoid matching inconsistencies across font versions and OS localizations, since font style names (e.g. "Bold") are frequently localized into many languages and the set of localizations available varies widely across platform and font version. User agents that match a concatenation of family name (nameID = 1) with style name (nameID = 2) are considered non-conformant.

This also allows for referencing faces that belong to larger families that cannot otherwise be referenced.

EXAMPLE 9

Use a local font or reference an SVG font in another document:

```
@font-face {
  font-family: Headline;
  src: local(Futura-Medium),
       url(fonts.svg#MyGeometricModern) format("svg");
}
```

Create an alias for local Japanese fonts on different platforms:

```
@font-face {
  font-family: jpgothic;
  src: local(HiraKakuPro-W3), local(Meiryo), local(IPAPGothic);
}
```

Reference a font face that cannot be matched within a larger family:

```
@font-face {
  font-family: Hoefler Text Ornaments;
  /* has the same font properties as Hoefler Text Regular */
  src: local(HoeflerText-Ornaments);
}
```

Since localized fullnames never match, a document with the header style rules below would always render using the default serif font, regardless whether a particular system locale parameter is set to Finnish or not:

```
@font-face {
  font-family: SectionHeader;
  src: local("Arial Lihavoitu");  /* Finnish fullname for Arial Bold, should fail */
  font-weight: bold;
}

h2 { font-family: SectionHeader, serif; }
```

A conformant user agent would never load the font ‘gentium.eot’ in the example below, since it is included in the first definition of the ‘src’ descriptor which is overridden by the second definition in the same `@font-face` rule:

```
@font-face {
  font-family: MainText;
  src: url(gentium.eot);                      /* for use with older user agents */
  src: local("Gentium"), url(gentium.woff);  /* Overrides src definition */
}
```

4.4 Font property descriptors: the font-style, font-weight, font-stretch descriptors

Name:	***font-style***
Value:	normal | italic | oblique
Initial:	normal

Name:	***font-weight***
Value:	normal | bold | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900
Initial:	normal

Name:	***font-stretch***

Value:	normal | ultra-condensed | extra-condensed | condensed | semi-condensed | semi-expanded | expanded | extra-expanded | ultra-expanded
Initial:	normal

These descriptors define the characteristics of a font face and are used in the process of matching styles to specific faces. For a font family defined with several `@font-face` rules, user agents can either download all faces in the family or use these descriptors to selectively download font faces that match actual styles used in document. The values for these descriptors are the same as those for the corresponding font properties except that relative keywords are not allowed, ‘bolder’ and ‘lighter’. If these descriptors are omitted, initial values are assumed.

The value for these font face style attributes is used in place of the style implied by the underlying font data. This allows authors to combine faces in flexible combinations, even in situations where the original font data was arranged differently. User agents that implement synthetic bolding and obliquing must only apply synthetic styling in cases where the font descriptors imply this is needed, rather than based on the style attributes implied by the font data.

4.5 Character range: the unicode-range descriptor

Name:	***unicode-range***
Value:	*<urange>* #
Initial:	U+0-10FFFF

This descriptor defines the set of Unicode codepoints that may be supported by the font face for which it is declared. The descriptor value is a comma-delimited list of Unicode range (*<urange>*) values. The union of these ranges defines the set of codepoints that serves as a hint for user agents when deciding whether or not to download a font resource for a given text run.

Each ***<urange>*** value is a `UNICODE-RANGE` token made up of a "U+" or "u+" prefix followed by a codepoint range in one of the three forms listed below. Ranges that do not fit one of the these forms are invalid and cause the declaration to be ignored.

single codepoint (e.g. U+416)
: a Unicode codepoint, represented as one to six hexadecimal digits

interval range (e.g. U+400-4ff)
: represented as two hyphen-separated Unicode codepoints indicating the inclusive start and end codepoints of a range

wildcard range (e.g. U+4??)
: defined by the set of codepoints implied when trailing ‘?’ characters signify any hexadeximal digit

Individual codepoints are written using hexadecimal values that correspond to Unicode character codepoints. Unicode codepoint values must be between 0 and 10FFFF inclusive. Digit values of codepoints are ASCII case-insensitive. For interval ranges, the start and end codepoints must be within the range noted above and the end codepoint must be greater than or equal to the start codepoint.

Wildcard ranges specified with ‘?’ that lack an initial digit (e.g. "U+???") are valid and equivalent to a wildcard range with an initial zero digit (e.g. "U+0???" = "U+0000-0FFF"). Wildcard ranges that extend beyond the range of Unicode codepoints are invalid. Because of this, the maximum number of trailing ‘?’ wildcard characters is five, even though the `UNICODE-RANGE` token accepts six.

Within the comma-delimited list of Unicode ranges in a ‘unicode-range’ descriptor declaration, ranges may overlap. The union of these ranges defines the set of codepoints for which the corresponding font may be used. User agents must not download or use the font for codepoints outside this set. User agents may normalize the list of ranges into a list that is different but represents the same set of codepoints.

The associated font might not contain glyphs for the entire set of codepoints defined by the ‘unicode-range’ descriptor. When the font is used, the ***effective character map*** is the intersection of the codepoints defined by ‘unicode-range’ with the font's *character map*. This allows authors to define supported ranges in terms of broad ranges without worrying about the precise codepoint ranges supported by the underlying font.

4.6 Using character ranges to define composite fonts

Multiple `@font-face` rules with different unicode ranges for the same family and style descriptor values can be used to create composite fonts that mix the glyphs from different fonts for different scripts. This can be used to combine fonts that only contain glyphs for a single script (e.g. Latin, Greek, Cyrillic) or it can be used by authors as a way of segmenting a font into fonts for commonly used characters and less frequently used characters. Since the user agent will only pull down the fonts it needs this helps reduce page bandwidth.

If the unicode ranges overlap for a set of `@font-face` rules with the same family and style descriptor values, the rules are ordered in the reverse order they were defined; the last rule defined is the first to be checked for a given character.

Example ranges for specific languages or characters:

unicode-range: U+A5;
: a single code point, the yen/yuan symbol

unicode-range: U+0-7F;
: code range for basic ASCII characters

unicode-range: U+590-5ff;
: code range for Hebrew characters

unicode-range: U+A5, U+4E00-9FFF, U+30??, U+FF00-FF9F;
: code range for Japanese kanji, hiragana and katakana characters plus yen/yuan symbol

EXAMPLE 10

The BBC provides news services in a wide variety of languages, many that are not well supported across all platforms. Using an `@font-face` rule, the BBC could provide a font for any of these languages, as it already does via a manual font download.

```
@font-face {
  font-family: BBCBengali;
  src: url(fonts/BBCBengali.woff) format("woff");
  unicode-range: U+00-FF, U+980-9FF;
}
```

EXAMPLE 11

Technical documents often require a wide range of symbols. The STIX Fonts project is one project aimed at providing fonts to support a wide range of technical typesetting in a standardized way. The example below shows the use of a font that provides glyphs for many of the mathematical and technical symbol ranges within Unicode:

```
@font-face {
  font-family: STIXGeneral;
  src: local(STIXGeneral), url(/stixfonts/STIXGeneral.otf);
  unicode-range: U+000-49F, U+2000-27FF, U+2900-2BFF, U+1D400-1D7FF;
}
```

EXAMPLE 12

This example shows how an author can override the glyphs used for Latin characters in a Japanese font with glyphs from a different font. The first rule specifies no range so it defaults to the entire range. The range specified in the second rule overlaps but takes precedence because it is defined later.

```
@font-face {
  font-family: JapaneseWithGentium;
  src: local(MSMincho);
  /* no range specified, defaults to entire range */
}

@font-face {
  font-family: JapaneseWithGentium;
  src: url(../fonts/Gentium.woff);
  unicode-range: U+0-2FF;
}
```

EXAMPLE 13

Consider a family constructed to optimize bandwidth by separating out Latin, Japanese and other characters into different font files:

```
/* fallback font - size: 4.5MB */
@font-face {
  font-family: DroidSans;
  src: url(DroidSansFallback.woff);
  /* no range specified, defaults to entire range */
}

/* Japanese glyphs - size: 1.2MB */
@font-face {
  font-family: DroidSans;
  src: url(DroidSansJapanese.woff);
  unicode-range: U+3000-9FFF, U+ff??;
}

/* Latin, Greek, Cyrillic along with some
   punctuation and symbols - size: 190KB */
@font-face {
  font-family: DroidSans;
  src: url(DroidSans.woff);
  unicode-range: U+000-5FF, U+1e00-1fff, U+2000-2300;
}
```

For simple Latin text, only the font for Latin characters is downloaded:

```
body { font-family: DroidSans; }

<p>This is that</p>
```

In this case the user agent first checks the unicode-range for the font containing Latin characters (DroidSans.woff). Since all the characters above are in the range U+0-5FF, the user agent downloads the font and renders the text with that font.

Next, consider text that makes use of an arrow character (⇨):

```
<p>This &#x21e8; that<p>
```

The user agent again first checks the unicode-range of the font containing Latin characters. Since U+2000-2300 includes the arrow code point (U+21E8), the user agent downloads the font. For this character however the Latin font does not have a matching glyph, so the effective unicode-range used for font matching excludes this code point. Next, the user agent evaluates the Japanese font. The unicode-range for the Japanese font, U+3000-9FFF and U+ff??, does not include U+21E8, so the user agent does not download the Japanese font. Next the fallback font is considered. The `@font-face` rule for the fallback font does not define unicode-range so its value defaults to the range of all Unicode code points. The fallback font is downloaded and used to render the arrow character.

4.7 Font features: the font-variant and font-feature-settings descriptors

Name:	***font-variant***
Value:	normal | none | [*<common-lig-values>* || *<discretionary-lig-values>* || *<historical-lig-values>* || *<contextual-alt-values>* || *stylistic(<feature-value-name>)* || *historical-forms* || *styleset(<feature-value-name> #)* || *character-variant(<feature-value-name> #)* || *swash(<feature-value-name>)* || *ornaments(<feature-value-name>)* || *annotation(<feature-value-name>)* || [small-caps | all-small-caps | petite-caps | all-petite-caps | unicase | titling-caps] || *<numeric-figure-values>* || *<numeric-spacing-values>* || *<numeric-fraction-values>* || ordinal || slashed-zero || *<east-asian-variant-values>* || *<east-asian-width-values>* || ruby]
Initial:	normal

Name:	***font-feature-settings***
Value:	normal | *<feature-tag-value>* #
Initial:	normal

These descriptors define initial settings that apply when the font defined by an `@font-face` rule is rendered. They do not affect font selection. Values are identical to those defined for the corresponding 'font-variant' and 'font-feature-settings' properties defined below except that the value 'inherit' is omitted. When multiple font feature descriptors or properties are used, the cumulative effect on text rendering is detailed in the section Font Feature Resolution below. In cases where specific values define synthesized fallback for certain 'font-variant' subproperties, the same synthesized fallback applies when used within those values are used with the 'font-variant' descriptor.

4.8 Font loading guidelines

The `@font-face` rule is designed to allow lazy loading of font resources that are only downloaded when used within a document. A stylesheet can include `@font-face` rules for a library of fonts of which only a select set are used; user agents must only download those fonts that are referred to within the style rules applicable to a given page. User agents that download all fonts defined in `@font-face` rules without considering whether those fonts are in fact used within a page are considered non-conformant. In cases where a font might be downloaded in character fallback cases, user agents may download a font if it's contained within the computed value of 'font-family' for a given text run.

```
@font-face {
  font-family: GeometricModern;
  src: url(font.woff);
}

p {
  /* font will be downloaded for pages with p elements */
  font-family: GeometricModern, sans-serif;
}

h2 {
  /* font may be downloaded for pages with h2 elements, even if Futura is available locally */
  font-family: Futura, GeometricModern, sans-serif;
}
```

In cases where textual content is loaded before downloadable fonts are available, user agents may render text as it would be rendered if downloadable font resources are not available or they may render text transparently with fallback fonts to avoid a flash of text using a fallback font. In cases where the font download fails user agents must display text, simply leaving transparent text is considered non-conformant behavior. Authors are advised to use fallback fonts in their font lists that closely match the metrics of the downloadable fonts to avoid large page reflows where possible.

4.9 Font fetching requirements

For font loads, user agents must use the potentially CORS-enabled fetch method defined by the [HTML5] specification for URL's defined within @font-face rules. When fetching, user agents must use "Anonymous" mode, set the referrer source to the stylesheet's URL and set the origin to the URL of the containing document.

> The implications of this for authors are that fonts will typically not be loaded cross-origin unless authors specifically takes steps to permit cross-origin loads. Sites can explicitly allow cross-site loading of font data using the `Access-Control-Allow-Origin` HTTP header. For other schemes, no explicit mechanism to allow cross-origin loading, beyond what is permitted by the potentially CORS-enabled fetch method, is defined or required.

EXAMPLE 14

For the examples given below, assume that a document is located at `http://example.com/page.html` and all URL's link to valid font resources supported by the user agent. Fonts defined with the ‘src’ descriptor values below will be loaded:

```
/* same origin (i.e. domain, scheme, port match document) */
src: url(fonts/simple.woff);

/* data url's with no redirects are treated as same origin */
src: url("data:application/font-woff;base64,...");

/* cross origin, different domain */
/* Access-Control-Allow-Origin response header set to '*' */
src: url(http://another.example.com/fonts/simple.woff);
```

Fonts defined with the ‘src’ descriptor values below will fail to load:

```
/* cross origin, different scheme */
/* no Access-Control-xxx headers in response */
src: url(https://example.com/fonts/simple.woff);

/* cross origin, different domain */
/* no Access-Control-xxx headers in response */
src: url(http://another.example.com/fonts/simple.woff);
```

5 Font Matching Algorithm

The algorithm below describes how fonts are associated with individual runs of text. For each character in the run a font family is chosen and a particular font face is selected containing a glyph for that character.

5.1 Case sensitivity of font family names

As part of the font matching algorithm outlined below, user agents must match font family names used in style rules with actual font family names contained in fonts available in a given environment or with font family names defined in `@font-face` rules. User agents must match these names case insensitively, using the "Default Caseless Matching" algorithm outlined in the Unicode specification [UNICODE]. This algorithm is detailed in section 3.13 entitled "Default Case Algorithms". Specifically, the algorithm must be applied without normalizing the strings involved and without applying any language-specific tailorings. The case folding method specified by this algorithm uses the case mappings with status field ‘C’ or ‘F’ in the CaseFolding.txt file of the Unicode Character Database.

> For authors this means that font family names are matched case insensitively, whether those names exist in a platform font or in the `@font-face` rules contained in a stylesheet. Authors should take care to ensure that names use a character sequence consistent with the actual font family name, particularly when using combining characters such as diacritical marks. For example, a family name that contains an uppercase A (U+0041) followed by a combining ring (U+030A) will **not** match a name that looks identical but which uses the precomposed lowercase a-ring character (U+00E5) instead of the combining sequence.

> Implementors should take care to verify that a given caseless string comparison implementation uses this precise algorithm and not assume that a given platform string matching routine follows it, as many of these have locale-specific behavior or use some level of string normalization.

5.2 Matching font styles

The procedure for choosing a font for a given character in a run of text consists of iterating over the font families named by the ‘font-family’ property, selecting a font face with the appropriate style based on other font properties and then determining whether a glyph exists for the given character. This is done using the ***character map*** of the font, data which maps characters to the default glyph for that character. A font is considered to ***support*** a given character if (1) the character is contained in the font's *character*

map and (2) if required by the containing script, shaping information is available for that character.

Some legacy fonts may include a given character in the *character map* but lack the shaping information (e.g. OpenType layout tables or Graphite tables) necessary for correctly rendering text runs containing that character.

Codepoint sequences consisting of a base character followed by a sequence of combining characters are treated slightly differently, see the section on cluster matching below.

For this procedure, the ***default face*** for a given font family is defined to be the face that would be selected if all font style properties were set to their initial value.

1. Using the computed font property values for a given element, the user agent starts with the first family name specified by the 'font-family' property.

2. If the family name is a generic family keyword, the user agent looks up the appropriate font family name to be used. User agents may choose the generic font family to use based on the language of the containing element or the Unicode range of the character.

3. For other family names, the user agent attempts to find the family name among fonts defined via `@font-face` rules and then among available system fonts, matching names with a case-insensitive comparison as outlined in the section above. On systems containing fonts with multiple localized font family names, user agents must match any of these names independent of the underlying system locale or platform API used. If the font resources defined for a given face in an `@font-face` rule are either not available or contain invalid font data, then the face should be treated as not present in the family. If no faces are present for a family defined via `@font-face` rules, the family should be treated as missing; matching a platform font with the same name must not occur in this case.

4. If a font family match occurs, the user agent assembles the set of font faces in that family and then narrows the set to a single face using other font properties in the order given below. A group of faces defined via `@font-face` rules with identical font descriptor values but differing 'unicode-range' values are considered to be a single ***composite face*** for this step:

    a. 'font-stretch' is tried first. If the matching set contains faces with width values matching the 'font-stretch' value, faces with other width values are removed from the matching set. If there is no face that exactly matches the width value the nearest width is used instead. If the value of 'font-stretch' is 'normal' or one of the condensed values, narrower width values are checked first, then wider values. If the value of 'font-stretch' is one of the expanded values, wider values are checked first, followed by narrower values. Once the closest matching width has been determined by this process, faces with other widths are removed from the matching set.

    b. 'font-style' is tried next. If the value of 'font-style' is 'italic', italic faces are checked first, then oblique, then normal faces. If the value is 'oblique', oblique faces are checked first, then italic faces and then normal faces. If the value is 'normal', normal faces are checked first, then oblique faces, then italic faces. Faces with other style values are excluded from the matching set. User agents are permitted to distinguish between italic and oblique faces within platform font families but this is not required, so all italic or oblique faces may be treated as italic faces. However, within font families defined via `@font-face` rules, italic and oblique faces must be distinguished using the value of the 'font-style' descriptor. For families that lack any italic or oblique faces, users agents may create artificial oblique faces, if this is permitted by the value of the 'font-synthesis' property.

    c. 'font-weight' is matched next, so it will always reduce the matching set to a single font face. If bolder/lighter relative weights are used, the effective weight is calculated based on the inherited weight value, as described in the definition of the 'font-weight' property. Given the desired weight and the weights of faces in the matching set after the steps above, if the desired weight is available that face matches. Otherwise, a weight is chosen using the rules below:

    - If the desired weight is less than 400, weights below the desired weight are checked in descending order followed by weights above the desired weight in ascending order until a match is found.
    - If the desired weight is greater than 500, weights above the desired weight are checked in ascending order followed by weights below the desired weight in descending order until a match is found.
    - If the desired weight is 400, 500 is checked first and then the rule for desired weights less than 400 is used.
    - If the desired weight is 500, 400 is checked first and then the rule for desired weights less than 400 is used.

d. ‘font-size’ must be matched within a UA-dependent margin of tolerance. (Typically, sizes for scalable fonts are rounded to the nearest whole pixel, while the tolerance for bitmapped fonts could be as large as 20%.) Further computations, e.g., by ‘em’ values in other properties, are based on the ‘font-size’ value that is used, not the one that is specified.

5. If the matched face is defined via `@font-face` rules, user agents must use the procedure below to select a single font:

   a. If the font resource has not been loaded and the range of characters defined by the ‘unicode-range’ descriptor value includes the character in question, load the font.

   b. After downloading, if the *effective character map* supports the character in question, select that font.

   When the matched face is a *composite face*, user agents must use the procedure above on each of the faces in the *composite face* in reverse order of `@font-face` rule definition.

   While the download occurs, user agents may either wait until the font is downloaded or render once with substituted font metrics and render again once the font is downloaded.

6. If no matching face exists or the matched face does not contain a glyph for the character to be rendered, the next family name is selected and the previous three steps repeated. Glyphs from other faces in the family are not considered. The only exception is that user agents may optionally substitute a synthetically obliqued version of the *default face* if that face supports a given glyph and synthesis of these faces is permitted by the value of the ‘font-synthesis’ property. For example, a synthetic italic version of the regular face may be used if the italic face doesn't support glyphs for Arabic.

7. If there are no more font families to be evaluated and no matching face has been found, then the user agent performs a ***system font fallback*** procedure to find the best match for the character to be rendered. The result of this procedure may vary across user agents.

8. If a particular character cannot be displayed using any font, the user agent should indicate by some means that a character is not being displayed, displaying either a symbolic representation of the missing glyph (e.g. using a Last Resort Font) or using the missing character glyph from a default font.

Optimizations of this process are allowed provided that an implementation behaves as if the algorithm had been followed exactly. Matching occurs in a well-defined order to ensure that the results are as consistent as possible across user agents, given an identical set of available fonts and rendering technology.

The ***first available font***, used in the definition of *font-relative lengths* such as ‘ex’ and ‘ch’, is defined to be the first available font that would match any character given font families in the ‘font-family’ list (or a user agent's default font if none are available).

5.3 Cluster matching

When text contains characters such as combining marks, ideally the base character should be rendered using the same font as the mark, this assures proper placement of the mark. For this reason, the font matching algorithm for clusters is more specialized than the general case of matching a single character by itself. For sequences containing variation selectors, which indicate the precise glyph to be used for a given character, user agents always attempt *system font fallback* to find the appropriate glyph before using the default glyph of the base character.

A sequence of codepoints containing combining mark or other modifiers is termed a grapheme cluster (see [CSS3TEXT] for a more complete description). For a given cluster containing a base character, *b* and a sequence of combining characters *c1, c2…*, the entire cluster is matched using these steps:

1. For each family in the font list, a face is chosen using the style selection rules defined in the previous section.
   a. If all characters in the sequence *b + c1 + c2 …* are completely supported by the font, select this font for the sequence.
   b. If a sequence of multiple codepoints is canonically equivalent to a single character and the font *supports* that character, select this font for the sequence and use the glyph associated with the canonically equiavlent character for the entire cluster.
2. If no font was found in the font list in step 1:
   a. If *c1* is a variation selector, system fallback must be used to find a font that *supports* the full sequence of *b + c1*. If no font on the system *supports* the full sequence, match the single character *b* using the normal procedure for matching single characters and ignore the variation selector. Note: a sequence with more than one variation

selector must be treated as an encoding error and the trailing selectors must be ignored. [UNICODE]

b. Otherwise, the user agent may optionally use system font fallback to match a font that *supports* the entire cluster.

3. If no font is found in step 2, use the matching sequence from step 1 to determine the longest sequence that is completely *supported* by a font in the font list and attempt to match the remaining combining characters separately using the rules for single characters.

5.4 Character handling issues

CSS font matching is always performed on text runs containing Unicode characters, so documents using legacy encodings are assumed to have been transcoded before matching fonts. For fonts containing *character maps* for both legacy encodings and Unicode, the contents of the legacy encoding *character map* must have no effect on the results of the font matching process.

The font matching process does not assume that text runs are in either normalized or denormalized form (see [CHARMOD-NORM] for more details). Fonts may only support precomposed forms and not the decomposed sequence of base character plus combining marks. Authors should always tailor their choice of fonts to their content, including whether that content contains normalized or denormalized character streams.

If a given character is a Private-Use Area Unicode codepoint, user agents must only match font families named in the 'font-family' list that are not generic families. If none of the families named in the 'font-family' list contain a glyph for that codepoint, user agents must display some form of missing glyph symbol for that character rather than attempting *system font fallback* for that codepoint. When matching the replacement character U+FFFD, user agents may skip the font matching process and immediately display some form of missing glyph symbol, they are not required to display the glyph from the font that would be selected by the font matching process.

In general, the fonts for a given family will all have the same or similar *character maps*. The process outlined here is designed to handle even font families containing faces with widely variant *character maps*. However, authors are cautioned that the use of such families can lead to unexpected results.

5.5 Font matching changes since CSS 2.1

The algorithm above is different from CSS 2.1 in a number of key places. These changes were made to better reflect actual font matching behavior across user agent implementations.

Differences compared to the font matching algorithm in CSS 2.1:

- The algorithm includes font-stretch matching.
- All possible font-style matching scenarios are delineated.
- Small-caps fonts are not matched as part of the font matching process, they are now handled via font features.
- Unicode variation selector matching is required.
- Cluster sequences are matched as a unit.

5.6 Font matching examples

EXAMPLE 15

It's useful to note that the CSS selector syntax may be used to create language-sensitive typography. For example, some Chinese and Japanese characters are unified to have the same Unicode code point, although the abstract glyphs are not the same in the two languages.

```
*:lang(ja) { font: 900 14pt/16pt "Heisei Mincho W9", serif; }
*:lang(zh-Hant-TW) { font: 800 14pt/16.5pt "Li Sung", serif; }
```

This selects any element that has the given language — Japanese or Traditional Chinese as used in Taiwan — and uses the appropriate font.

6 Font Feature Properties

Modern font technologies support a variety of advanced typographic and language-specific font features. Using these features, a single font can provide glyphs for a wide range of ligatures, contextual and stylistic alternates, tabular and old-style figures, small capitals, automatic fractions, swashes, and alternates specific to a given language. To allow authors control over these font capabilities, the ‘font-variant’ property has been expanded for CSS3. It now functions as a shorthand for a set of properties that provide control over stylistic font features.

6.1 Glyph selection and positioning

Simple fonts used for displaying Latin text use a very basic processing model. Fonts contain a *character map* which maps each character to a glyph for that character. Glyphs for subsequent characters are simply placed one after the other along a run of text. Modern font formats such as OpenType and AAT (Apple Advanced Typography) use a richer processing model. The glyph for a given character can be chosen and positioned not just based on the codepoint of the character itself, but also on adjacent characters as well as the language, script, and features enabled for the text. Font features may be required for specific scripts, or recommended as enabled by default or they might be stylistic features meant to be used under author control.

For a good visual overview of these features, see the [OPENTYPE-FONT-GUIDE]. For a detailed description of glyph processing for OpenType fonts, see [WINDOWS-GLYPH-PROC].

Stylistic font features can be classified into two broad categories: ones that affect the harmonization of glyph shapes with the surrounding context, such as kerning and ligature features, and ones such as the small-caps, subscript/superscript and alternate features that affect shape selection.

The subproperties of ‘font-variant’ listed below are used to control these stylistic font features. They do not control features that are required for displaying certain scripts, such as the OpenType features used when displaying Arabic or Indic language text. They affect glyph selection and positioning, but do not affect font selection as described in the font matching section (except in cases required for compatibility with CSS 2.1).

To assure consistent behavior across user agents, the equivalent OpenType property settings are listed for individual properties and are normative. When using other font formats these should be used as a guideline to map CSS font feature property values to specific font features.

6.2 Language-specific display

OpenType also supports language-specific glyph selection and positioning, so that text can be displayed correctly in cases where the language dictates a specific display behavior. Many languages share a common script, but the shape of certain letters can vary across those languages. For example, certain Cyrillic letters have different shapes in Russian text than in Bulgarian. In Latin text, it's common to render "fi" with an explicit fi-ligature that lacks a dot on the "i". However, in languages such as Turkish which uses both a dotted-i and a dotless-i, it's important to not use this ligature or use a specialized version that contains a dot over the "i". The example below shows language-specific variations based on stylistic traditions found in Spanish, Italian and French orthography:

If the content language of the element is known according to the rules of the document language, user agents are required to infer the OpenType language system from the content language and use that when selecting and positioning glyphs using an OpenType font.

For OpenType fonts, in some cases it may be necessary to explicitly declare the OpenType language to be used, for example when displaying text in a given language that uses the typographic conventions of another language or when the font does not explicitly support a given language but supports a language that shares common typographic conventions. The 'font-language-override' property is used for this purpose.

6.3 Kerning: the font-kerning property

Name:	***font-kerning***
Value:	auto | normal | none
Initial:	auto
Applies to:	all elements
Inherited:	yes
Percentages:	N/A
Media:	visual
Computed value:	as specified
Animatable:	no

Kerning is the contextual adjustment of inter-glyph spacing. This property controls metric kerning, kerning that utilizes adjustment data contained in the font.

auto
: Specifies that kerning is applied at the discretion of the user agent

normal
: Specifies that kerning is applied

none
: Specifies that kerning is not applied

For fonts that do not include kerning data this property will have no visible effect. When rendering with OpenType fonts, the [OPENTYPE] specification suggests that kerning be enabled by default. When kerning is enabled, the OpenType `kern` feature is enabled (for vertical text runs the `vkrn` feature is enabled instead). User agents must also support fonts that only support kerning via data contained in a `kern` font table, as detailed in the OpenType specification. If the 'letter-spacing' property is defined, kerning adjustments are considered part of the default spacing and letter spacing adjustments are made after kerning has been applied.

When set to 'auto', user agents can determine whether to apply kerning or not based on a number of factors: text size, script, or other factors that influence text processing speed. Authors who want proper kerning should use 'normal' to explicitly enable kerning. Likewise, some authors may prefer to disable kerning in situations where performance is more important than precise appearance. However, in well-designed modern implementations the use of kerning generally does not have a large impact on

text rendering speed.

6.4 Ligatures: the font-variant-ligatures property

Name:	***font-variant-ligatures***
Value:	normal | none | [*<common-lig-values>* || *<discretionary-lig-values>* || *<historical-lig-values>* || *<contextual-alt-values>*]
Initial:	normal
Applies to:	all elements
Inherited:	yes
Percentages:	N/A
Media:	visual
Computed value:	as specified
Animatable:	no

Ligatures and contextual forms are ways of combining glyphs to produce more harmonized forms.

```
<common-lig-values>        = [ common-ligatures | no-common-ligatures ]
<discretionary-lig-values> = [ discretionary-ligatures | no-discretionary-ligatures ]
<historical-lig-values>    = [ historical-ligatures | no-historical-ligatures ]
<contextual-alt-values>    = [ contextual | no-contextual ]
```

Individual values have the following meanings:

normal

A value of 'normal' specifies that common default features are enabled, as described in detail in the next section. For OpenType fonts, common ligatures and contextual forms are on by default, discretionary and historical ligatures are not.

none

Specifies that all types of ligatures and contextual forms covered by this property are explicitly disabled. In situations where ligatures are not considered necessary, this may improve the speed of text rendering.

common-ligatures

Enables display of common ligatures (OpenType features: `liga, clig`). For OpenType fonts, common ligatures are enabled by default.

no-common-ligatures

Disables display of common ligatures (OpenType features: `liga, clig`).

discretionary-ligatures

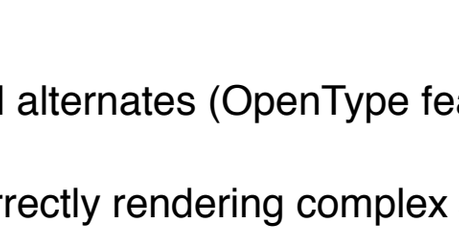
Enables display of discretionary ligatures (OpenType feature: `dlig`). Which ligatures are discretionary or optional is decided by the type designer, so authors will need to refer to the documentation of a given font to understand which ligatures are considered discretionary.

no-discretionary-ligatures

Disables display of discretionary ligatures (OpenType feature: `dlig`).

historical-ligatures

Enables display of historical ligatures (OpenType feature: `hlig`).



no-historical-ligatures

Disables display of historical ligatures (OpenType feature: `hlig`).

contextual

Enables display of contextual alternates (OpenType feature: `calt`). Although not strictly a ligature feature, like ligatures this feature is commonly used to harmonize the shapes of glyphs with the surrounding context. For OpenType fonts, this feature is on by default.

no-contextual

Disables display of contextual alternates (OpenType feature: `calt`).

Required ligatures, needed for correctly rendering complex scripts, are not affected by the settings above, including ‘none’ (OpenType feature: `rlig`).

6.5 Subscript and superscript forms: the font-variant-position property

Name:	***font-variant-position***
Value:	normal | sub | super
Initial:	normal
Applies to:	all elements
Inherited:	yes
Percentages:	N/A
Media:	visual
Computed value:	as specified
Animatable:	no

This property is used to enable typographic subscript and superscript glyphs. These are alternate glyphs designed within the same em-box as default glyphs and are intended to be laid out on the same baseline as the default glyphs, with no resizing or repositioning of the baseline. They are explicitly designed to match the surrounding text and to be more readable without affecting the line height.

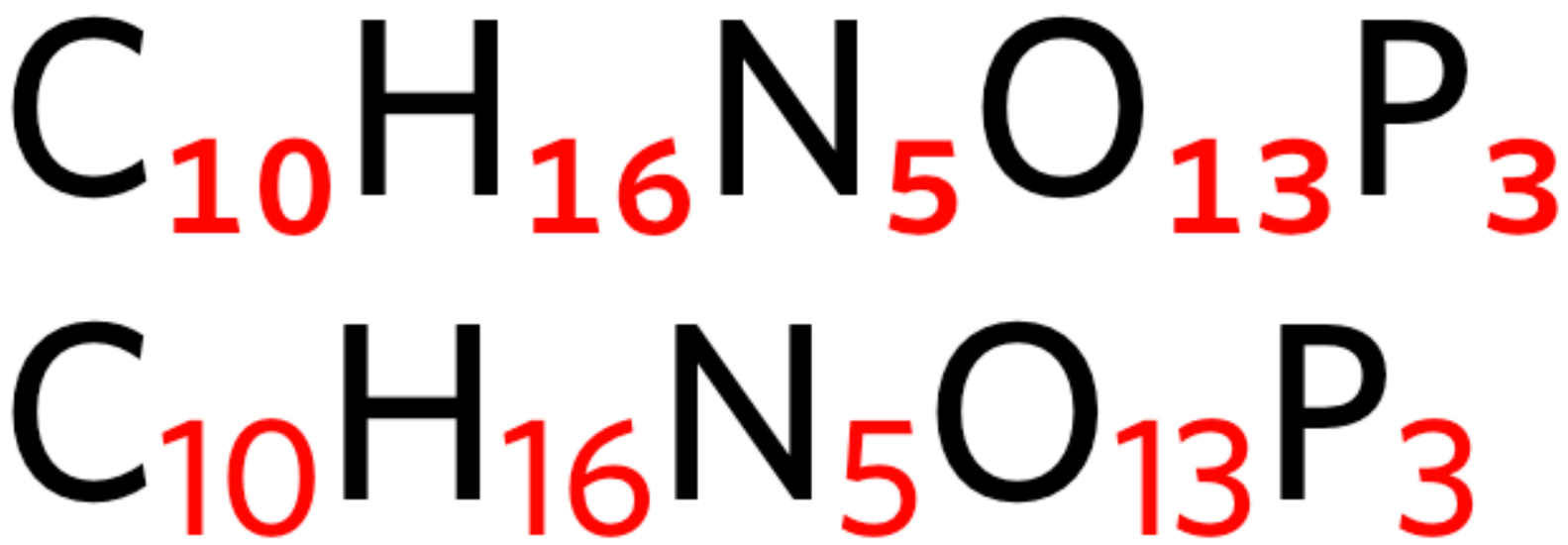


Figure 21. *Subscript glyphs (top) vs. typical synthesized subscripts (bottom)*

Individual values have the following meanings:

normal
: None of the features listed below are enabled.

sub
: Enables display of subscript variants (OpenType feature: `subs`).

super
: Enables display of superscript variants (OpenType feature: `sups`).

Because of the semantic nature of subscripts and superscripts, when the value is either ‘sub’ or ‘super’ for a given contiguous run of text, if a variant glyph is not available for all the characters in the run, simulated glyphs must be synthesized for all characters using reduced forms of the glyphs that would be used without this feature applied. This is done per run to avoid a mixture of variant glyphs and synthesized ones that would not align correctly. In the case of OpenType fonts that lack subscript or superscript glyphs for a given character, user agents must use the appropriate subscript and superscript metrics specified in the selected font's OS/2 table [OPENTYPE] to calculate the size and offset of the synthesized substitutes.

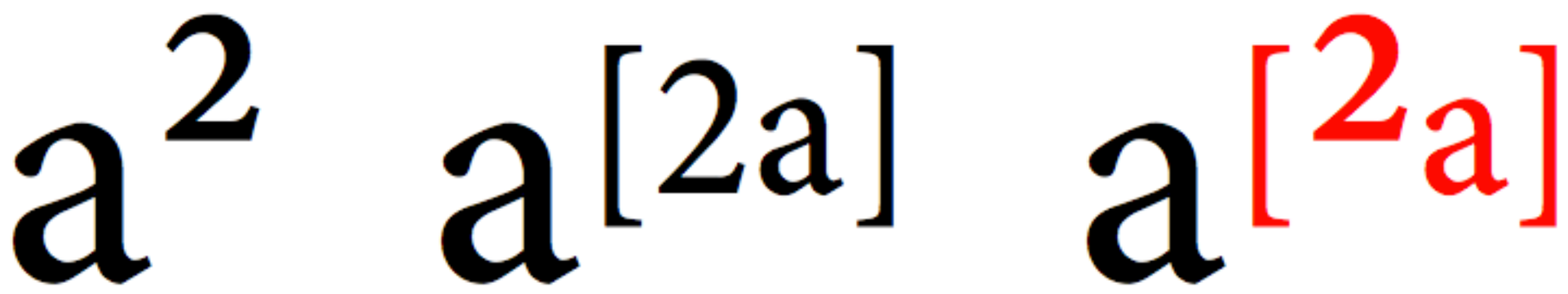


Figure 22. *Superscript alternate glyph (left), synthesized superscript glyphs (middle), and incorrect mixture of the two (right)*

In situations where text decorations are only applied to runs of text containing superscript or subscript glyphs, the synthesized glyphs must be used to avoid problems with the placement of decorations.

In the past, user agents have used font-size and vertical-align to simulate subscripts and superscripts for the `sub` and `sup` elements. To allow a backwards compatible way of defining subscripts and superscripts, it is recommended that authors use conditional rules [CSS3-CONDITIONAL] so that older user agents will still render subscripts and superscripts via the older mechanism.

Authors should note that fonts typically only provide subscript and superscript glyphs for a subset of all characters supported by the font. For example, while subscript and superscript glyphs are often available for Latin numbers, glyphs for punctuation and letter characters are less frequently provided. The synthetic fallback rules defined for this property assure that subscripts and superscripts will always appear but the appearance may not match author expectations if the font used does not provide the appropriate alternate glyph for all characters contained in a subscript or superscript.

This property is not cumulative. Applying it to elements within a subscript or superscript won't nest the placement of a subscript or superscript glyph. Images contained within text runs where the value of this property is ‘sub’ or ‘super’ will be drawn just as they would if the value was ‘normal’.

Because of these limitations, ‘font-variant-position’ is not recommended for use in user agent stylesheets. Authors should use it in

cases where subscripts or superscripts will only contain the narrow range of characters supported by the fonts specified.

The variant glyphs use the same baseline as the default glyphs would use. There is no shift in the placement along the baseline, so the use of variant glyphs doesn't affect the height of the inline box or alter the height of the linebox. This makes superscript and subscript variants ideal for situations where it's important that leading remain constant, such as in multi-column layout.

EXAMPLE 16

A typical user agent default style for the sub element:

```
sub {
  vertical-align: sub;
  font-size: smaller;
  line-height: normal;
}
```

Using font-variant-position to specify typographic subscripts in a way that will still show subscripts in older user agents:

```
@supports ( font-variant-position: sub ) {

  sub {
    vertical-align: baseline;
    font-size: 100%;
    line-height: inherit;
    font-variant-position: sub;
  }

}
```

User agents that support the ‘font-variant-position’ property will select a subscript variant glyph and render this without adjusting the baseline or font-size. Older user agents will ignore the ‘font-variant-position’ property definition and use the standard defaults for subscripts.

6.6 Capitalization: the font-variant-caps property

Name:	***font-variant-caps***
Value:	normal | small-caps | all-small-caps | petite-caps | all-petite-caps | unicase | titling-caps
Initial:	normal
Applies to:	all elements
Inherited:	yes
Percentages:	N/A
Media:	visual
Computed value:	as specified
Animatable:	no

This property allows the selection of alternate glyphs used for small or petite capitals or for titling. These glyphs are specifically designed to blend well with the surrounding normal glyphs, to maintain the weight and readability which suffers when text is simply resized to fit this purpose.

Individual values have the following meanings:

normal

None of the features listed below are enabled.

small-caps

Enables display of small capitals (OpenType feature: `smcp`). Small-caps glyphs typically use the form of uppercase letters but are reduced to the size of lowercase letters.

all-small-caps

Enables display of small capitals for both upper and lowercase letters (OpenType features: `c2sc, smcp`).

petite-caps

Enables display of petite capitals (OpenType feature: `pcap`).

all-petite-caps

Enables display of petite capitals for both upper and lowercase letters (OpenType features: `c2pc, pcap`).

unicase

Enables display of mixture of small capitals for uppercase letters with normal lowercase letters (OpenType feature: `unic`).

titling-caps

Enables display of titling capitals (OpenType feature: `titl`). Uppercase letter glyphs are often designed for use with lowercase letters. When used in all uppercase titling sequences they can appear too strong. Titling capitals are designed specifically for this situation.

The availability of these glyphs is based on whether a given feature is defined or not in the feature list of the font. User agents can optionally decide this on a per-script basis but should explicitly not decide this on a per-character basis.

Some fonts may only support a subset or none of the features described for this property. For backwards compatibility with CSS 2.1, if ‘small-caps’ or ‘all-small-caps’ is specified but small-caps glyphs are not available for a given font, user agents should simulate a small-caps font, for example by taking a normal font and replacing the glyphs for lowercase letters with scaled versions of the glyphs for uppercase characters (replacing the glyphs for both upper and lowercase letters in the case of ‘all-small-caps’).

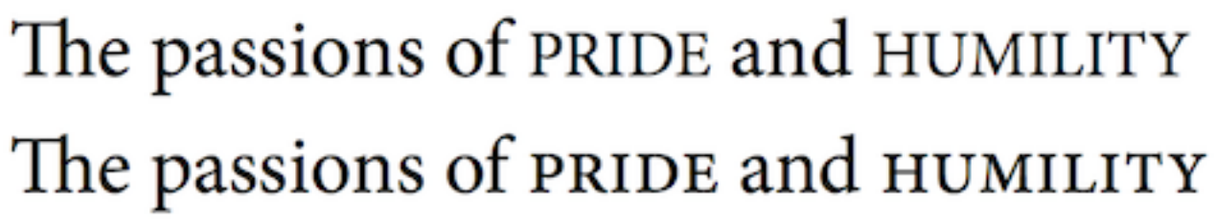


Figure 23. *Synthetic vs. real small-caps*

To match the surrounding text, a font may provide alternate glyphs for caseless characters when these features are enabled but when a user agent simulates small capitals, it must not attempt to simulate alternates for codepoints which are considered caseless.

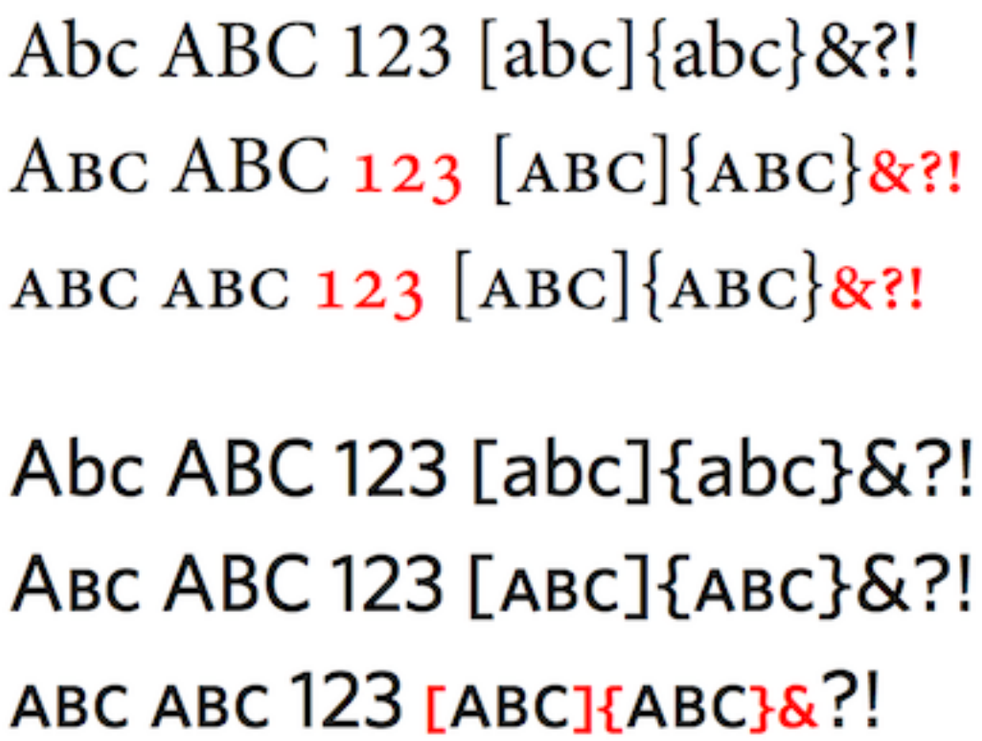


Figure 24. *Caseless characters with small-caps, all-small-caps enabled*

If either ‘petite-caps’ or ‘all-petite-caps’ is specified for a font that doesn't support these features, the property behaves as if

'small-caps' or 'all-small-caps', respectively, had been specified. If 'unicase' is specified for a font that doesn't support that feature, the property behaves as if 'small-caps' was applied only to lowercased uppercase letters. If 'titling-caps' is specified with a font that does not support this feature, this property has no visible effect. When simulated small capital glyphs are used, for scripts that lack uppercase and lowercase letters, 'small-caps', 'all-small-caps', 'petite-caps', 'all-petite-caps' and 'unicase' have no visible effect.

When casing transforms are used to simulate small capitals, the casing transformations must match those used for the 'text-transform' property.

As a last resort, unscaled uppercase letter glyphs in a normal font may replace glyphs in a small-caps font so that the text appears in all uppercase letters.

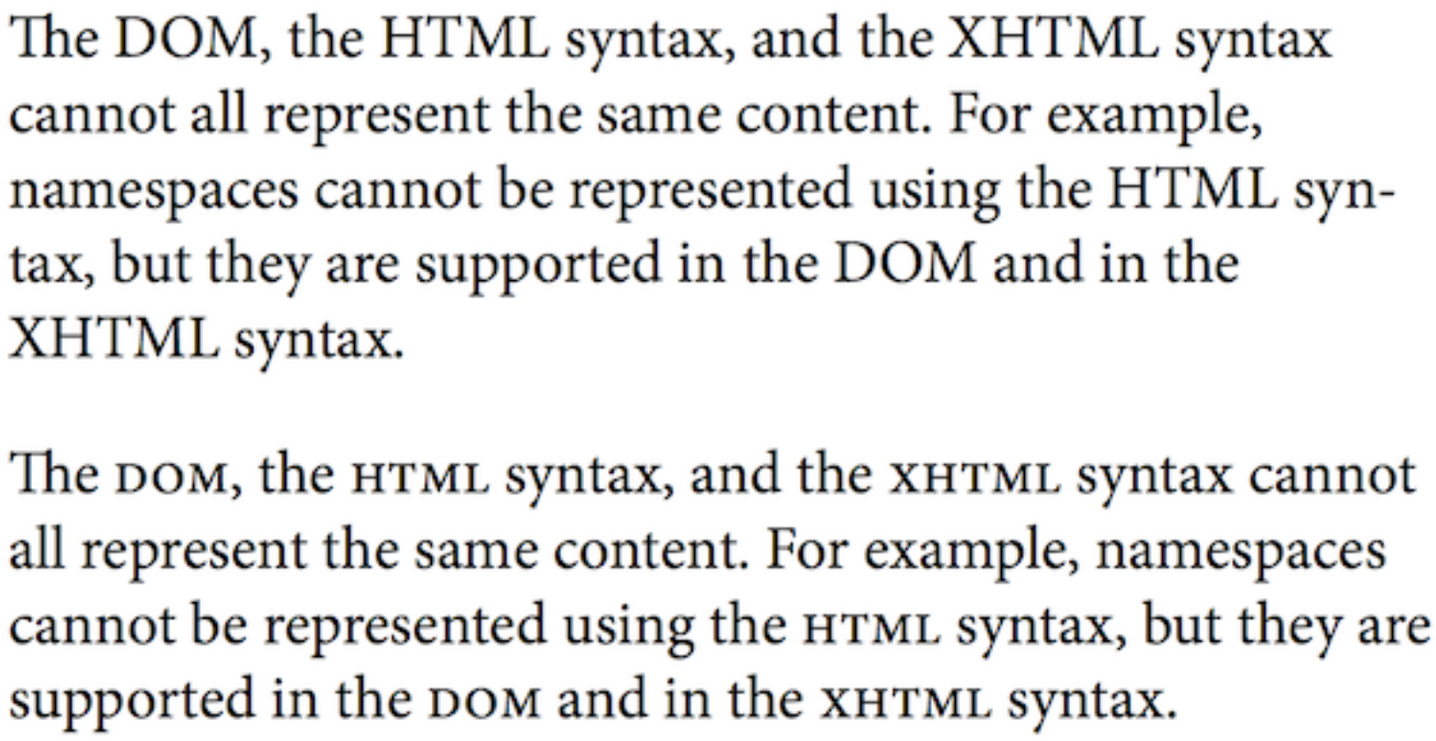


Figure 25. *Using small capitals to improve readability in acronym-laden text*

EXAMPLE 17

Quotes rendered italicised, with small-caps on the first line:

```
blockquote            { font-style: italic; }
blockquote:first-line { font-variant: small-caps; }

<blockquote>I'll be honor-bound to slap them like a haddock.</blockquote>
```

6.7 Numerical formatting: the font-variant-numeric property

Name:	***font-variant-numeric***
Value:	normal | [*<numeric-figure-values>* || *<numeric-spacing-values>* || *<numeric-fraction-values>* || ordinal || slashed-zero]
Initial:	normal
Applies to:	all elements
Inherited:	yes
Percentages:	N/A
Media:	visual
Computed value:	as specified
Animatable:	no

Specifies control over numerical forms. The example below shows how some of these values can be combined to influence the rendering of tabular data with fonts that support these features. Within normal paragraph text, proportional numbers are used

while tabular numbers are used so that columns of numbers line up properly:

	Lining	Old-Style
Proportional	409,280	409,280
	367,112	367,112
	155,068	155,068
	171,792	171,792
Tabular	409,280	409,280
	367,112	367,112
	155,068	155,068
	171,792	171,792

Figure 26. *Using number styles*

Possible combinations:

<numeric-figure-values> = [`lining-nums` | `oldstyle-nums`]

<numeric-spacing-values> = [`proportional-nums` | `tabular-nums`]

<numeric-fraction-values> = [`diagonal-fractions` | `stacked-fractions`]

Individual values have the following meanings:

normal
: None of the features listed below are enabled.

lining-nums
: Enables display of lining numerals (OpenType feature: `lnum`).

oldstyle-nums
: Enables display of old-style numerals (OpenType feature: `onum`).

proportional-nums
: Enables display of proportional numerals (OpenType feature: `pnum`).

tabular-nums
: Enables display of tabular numerals (OpenType feature: `tnum`).

diagonal-fractions
: Enables display of lining diagonal fractions (OpenType feature: `frac`).

2 1/3 ▶ 2⅓

stacked-fractions
: Enables display of lining stacked fractions (OpenType feature: `afrc`).

2 1/3 ▶ $2\frac{1}{3}$

ordinal
: Enables display of letter forms used with ordinal numbers (OpenType feature: `ordn`).

1st 17th 2a ▶ 1st 17th 2a

slashed-zero

Enables display of slashed zeros (OpenType feature: `zero`).

EXAMPLE 18

In the case of 'ordinal', although ordinal forms are often the same as superscript forms, they are marked up differently.

For superscripts, the variant property is only applied to the sub-element containing the superscript:

```
sup { font-variant-position: super; }
x<sup>2</sup>
```

For ordinals, the variant property is applied to the entire ordinal number rather than just to the suffix (or to the containing paragraph):

```
.ordinal { font-variant-numeric: ordinal; }
<span class="ordinal">17th</span>
```

In this case only the "th" will appear in ordinal form, the digits will remain unchanged. Depending upon the typographic traditions used in a given language, ordinal forms may differ from superscript forms. In Italian, for example, ordinal forms sometimes include an underline in the ordinal design.

EXAMPLE 19

A simple flank steak marinade recipe, rendered with automatic fractions and old-style numerals:

```
.amount { font-variant-numeric: oldstyle-nums diagonal-fractions; }

<h4>Steak marinade:</h4>
<ul>
  <li><span class="amount">2</span> tbsp olive oil</li>
  <li><span class="amount">1</span> tbsp lemon juice</li>
  <li><span class="amount">1</span> tbsp soy sauce</li>
  <li><span class="amount">1 1/2</span> tbsp dry minced onion</li>
  <li><span class="amount">2 1/2</span> tsp italian seasoning</li>
  <li>Salt &amp; pepper</li>
</ul>

<p>Mix the meat with the marinade and let it sit covered in the refrigerator
for a few hours or overnight.</p>
```

Note that the fraction feature is only applied to values not the entire paragraph. Fonts often implement this feature using contextual rules based on the use of the slash ('/') character. As such, it's not suitable for use as a paragraph-level style.

6.8 Alternates and swashes: the font-variant-alternates property

Name:	***font-variant-alternates***
Value:	normal | [stylistic(<feature-value-name>) || historical-forms || styleset(<feature-value-name> #) || character-variant(<feature-value-name> #) || swash(<feature-value-name>) || ornaments(<feature-value-name>) || annotation(<feature-value-name>)]
Initial:	normal
Applies to:	all elements
Inherited:	yes
Percentages:	N/A
Media:	visual
Computed value:	as specified
Animatable:	no

For any given character, fonts can provide a variety of alternate glyphs in addition to the default glyph for that character. This property provides control over the selection of these alternate glyphs.

For many of the property values listed below, several different alternate glyphs are available. How many alternates are available and what they represent is font-specific, so these are each marked ***font specific*** in the value definitions below. Because the nature of these alternates is font-specific, the `@font-feature-values` rule is used to define values for a specific font family or set of families that associate a font-specific numeric `<feature-index>` with a custom `<feature-value-name>`, which is then used in this property to select specific alternates:

```
@font-feature-values Noble Script { @swash { swishy: 1; flowing: 2; } }

p {
  font-family: Noble Script;
  font-variant-alternates: swash(flowing); /* use swash alternate #2 */
}
```

When a particular `<feature-value-name>` has not been defined for a given family or for a particular feature type, the computed value must be the same as if it had been defined. However, property values that contain these undefined `<feature-value-name>` identifiers must be ignored when choosing glyphs.

```
/* these two style rules are effectively the same */
p { font-variant-alternates: swash(unknown-value); } /* not a defined value, ignored */
p { font-variant-alternates: normal; }
```

This allows values to be defined and used for a given set of font families but ignored if fallback occurs, since the font family name would be different. If a given value is outside the range supported by a given font, the value is ignored. These values never apply to generic font families.

Individual values have the following meanings:

normal

None of the features listed below are enabled.

historical-forms

Enables display of historical forms (OpenType feature: `hist`).

stylistic(<feature-value-name>)

Enables display of stylistic alternates (*font specific*, OpenType feature: `salt <feature-index>`).

styleset(<feature-value-name> #)

Enables display with stylistic sets (*font specific*, OpenType feature: `ss<feature-index>` OpenType currently defines `ss01` through `ss20`).

character-variant(<feature-value-name> #)

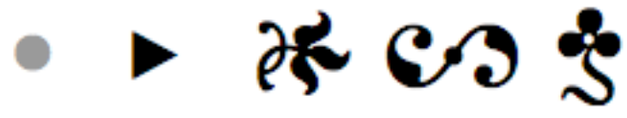
Enables display of specific character variants (*font specific*, OpenType feature: `cv<feature-index>` OpenType currently defines `cv01` through `cv99`).

swash(<feature-value-name>)

Enables display of swash glyphs (*font specific*, OpenType feature: `swsh <feature-index>, cswh <feature-index>`).

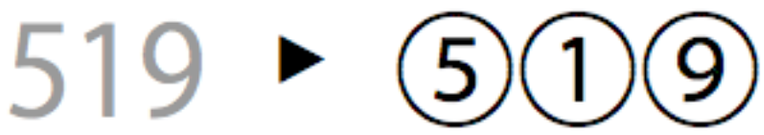
ornaments(<feature-value-name>)

Enables replacement of default glyphs with ornaments, if provided in the font (*font specific*, OpenType feature: `ornm <feature-index>`). Some fonts may offer ornament glyphs as alternates for a wide collection of characters; however, displaying arbitrary characters (e.g., alphanumerics) as ornaments is poor practice as it distorts the semantics of the data. Font designers are encouraged to encode all ornaments (except those explicitly encoded in the Unicode Dingbats blocks, etc.) as alternates for the bullet character (U+2022) to allow authors to select the desired glyph using ‘ornaments’.



annotation(<feature-value-name>)

Enables display of alternate annotation forms (*font specific*, OpenType feature: `nalt <feature-index>`).



6.9 Defining font specific alternates: the `@font-feature-values` rule

Several of the possible values of ‘font-variant-alternates’ listed above are labeled as *font specific*. For these features fonts may define not just a single glyph but a set of alternate glyphs with an index to select a given alternate. Since these are font family specific, the `@font-feature-values` rule is used to define named values for these indices for a given family.

> See the object model reference section for a description of the interfaces used to modify these rules via the CSS Object Model.

EXAMPLE 20

In the case of the swash Q in the example shown above, the swash could be specified using these style rules:

```
@font-feature-values Jupiter Sans {
  @swash {
    delicate: 1;
    flowing: 2;
  }
}

h2 { font-family: Jupiter Sans, sans-serif; }

/* show the second swash variant in h2 headings */
h2:first-letter { font-variant-alternates: swash(flowing); }

<h2>Quick</h2>
```

When Jupiter Sans is present, the second alternate swash alternate will be displayed. When not present, no swash character will be shown, since the specific named value "flowing" is only defined for the Jupiter Sans family. The @-mark indicates the name of the property value for which a named value can be used. The name "flowing" is chosen by the author. The index that represents each alternate is defined within a given font's data.

6.9.1 Basic syntax

An `@font-feature-values` rule is composed of a list of font families followed by a block containing individual feature value blocks that take the form of @-rules. Each block defines a set of named values for a specific font feature when a given set of font families is used. Effectively, they define a mapping of ⟨family, feature, ident⟩ → ⟨values⟩ where ⟨values⟩ are the numeric indices used for specific features defined for a given font.

In terms of the grammar, this specification defines the following productions:

```
font_feature_values_rule
  : FONT_FEATURE_VALUES_SYM S* font_family_name_list S*
    '{' S* feature_value_block? [ S* feature_value_block? ]* '}' S*
  ;

font_family_name_list
  : font_family_name [ S* ',' S* font_family_name ]*
  ;

font_family_name
  : STRING | [ IDENT [ S* IDENT ]* ]
  ;

feature_value_block
  : feature_type S*
    '{' S* feature_value_definition? [ S* ';' S* feature_value_definition? ]* '}' S*
  ;

feature_type:
  ATKEYWORD
  ;

feature_value_definition
  : IDENT S* ':' S* NUMBER [ S* NUMBER ]*
  ;
```

The following new token is introduced:

```
@{F}{O}{N}{T}{-}{F}{E}{A}{T}{U}{R}{E}{-}{V}{A}{L}{U}{E}{S}   {return FONT_FEATURE_VALUES_SYM;}
```

Feature value blocks are handled as at-rules, they consist of everything up to the next block or semi-colon, whichever comes first.

The font family list is a comma-delimited list of font family names that match the definition of *<family-name>* for the 'font-family' property. This means that only named font families are allowed, rules that include generic or system fonts in the list of font families are syntax errors. However, if a user agent defines a generic font to be a specific named font (e.g. Helvetica), the settings associated with that family name will be used. If syntax errors occur within the font family list, the entire rule must be ignored.

Within feature value blocks, the feature type is '@' followed by the name of one of the *font specific* property values of 'font-variant-alternates' (e.g. `@swash`). The identifiers used within feature value definitions follow the rules of CSS user identifiers and are case-sensitive. They are unique only for a given set of font families and feature type. The same identifier used with a different feature type is treated as a separate and distinct value. If the same identifier is defined mulitple times for a given feature type and font family, the last defined value is used. Values associated with a given identifier are limited to integer values 0 or greater.

When syntax errors occur within a feature value definition, such as invalid identifiers or values, the entire feature value definition must be omitted, just as syntax errors in style declarations are handled. When the feature type is invalid, the entire associated feature value block must be ignored.

EXAMPLE 21

Rules that are equivalent given syntax error handling:

```
@font-feature-values Bongo {
  @swash { ornate: 1; }
  annotation { boxed: 4; } /* should be @annotation! */
  @swash { double-loops: 1; flowing: -1; } /* negative value */
  @ornaments ; /* incomplete definition */
  @styleset { double-W: 14; sharp-terminals: 16 1 } /* missing ; */
  redrum  /* random editing mistake */
}
```

The example above is equivalent to:

```
@font-feature-values Bongo {
  @swash { ornate: 1; }
  @swash { double-loops: 1; }
  @styleset { double-W: 14; sharp-terminals: 16 1; }
}
```

If multiple `@font-feature-values` rules are defined for a given family, the resulting values definitions are the union of the definitions contained within these rules. This allows a set of named values to be defined for a given font family globally for a site and specific additions made per-page.

EXAMPLE 22

Using both site-wide and per-page feature values:

```
site.css:

  @font-feature-values Mercury Serif {
    @styleset {
      stacked-g: 3; /* "two-storey" versions of g, a */
      stacked-a: 4;
    }
  }

page.css:

  @font-feature-values Mercury Serif {
    @styleset {
      geometric-m: 7; /* alternate version of m */
    }
  }

  body {
    font-family: Mercury Serif, serif;

    /* enable both the use of stacked g and alternate m */
    font-variant-alternates: styleset(stacked-g, geometric-m);
  }
```

EXAMPLE 23

Using a commonly named value allows authors to use a single style rule to cover a set of fonts for which the underlying selector is different for each font. If either font in the example below is found, a circled number glyph will be used:

```
@font-feature-values Taisho Gothic {
  @annotation { boxed: 1; circled: 4; }
}

@font-feature-values Otaru Kisa {
  @annotation { circled: 1; black-boxed: 3; }
}

h3.title {
  /* circled form defined for both fonts */
  font-family: Taisho Gothic, Otaru Kisa;
  font-variant: annotation(circled);
}
```

6.9.2 Multi-valued feature value definitions

Most *font specific* ‘font-variant-alternates’ property values take a single value (e.g. ‘swash’). The ‘character-variant’ property value allows two values and ‘styleset’ allows an unlimited number.

For the styleset property value, multiple values indicate the style sets to be enabled. Values between 1 and 99 enable OpenType features `ss01` through `ss99`. However, the OpenType standard only officially defines `ss01` through `ss20`. For OpenType fonts, values greater than 99 or equal to 0 do not generate a syntax error when parsed but enable no OpenType features.

```
@font-feature-values Mars Serif {
  @styleset {
    alt-g: 1;        /* implies ss01 = 1 */
    curly-quotes: 3; /* implies ss03 = 1 */
    code: 4 5;       /* implies ss04 = 1, ss05 = 1 */
  }

  @styleset {
    dumb: 125;        /* >99, ignored */
  }

  @swash {
    swishy: 3 5;     /* more than 1 value for swash, syntax error */
  }
}

p.codeblock {
  /* implies ss03 = 1, ss04 = 1, ss05 = 1 */
  font-variant-alternates: styleset(curly-quotes, code);
}
```

For character-variant, a single value between 1 and 99 indicates the enabling of OpenType feature `cv01` through `cv99`. For OpenType fonts, values greater than 99 or equal to 0 are ignored but do not generate a syntax error when parsed but enable no OpenType features. When two values are listed, the first value indicates the feature used and the second the value passed for that feature. If more than two values are assigned to a given name, a syntax error occurs and the entire feature value definition is ignored.

```
@font-feature-values MM Greek {
  @character-variant { alpha-2: 1 2; }   /* implies cv01 = 2 */
  @character-variant { beta-3: 2 3; }    /* implies cv02 = 3 */
  @character-variant { epsilon: 5 3 6; } /* more than 2 values, syntax error, definition ignored */
  @character-variant { gamma: 12; }      /* implies cv12 = 1 */
  @character-variant { zeta:   20 3; }   /* implies cv20 = 3 */
  @character-variant { zeta-2: 20 2; }   /* implies cv20 = 2 */
  @character-variant { silly: 105; }     /* >99, ignored */
  @character-variant { dumb: 323 3; }    /* >99, ignored */
}

#title {
  /* use the third alternate beta, first alternate gamma */
  font-variant-alternates: character-variant(beta-3, gamma);
}

p {
  /* zeta-2 follows zeta, implies cv20 = 2  */
  font-variant-alternates: character-variant(zeta, zeta-2);
}

.special {
  /* zeta follows zeta-2, implies cv20 = 3  */
  font-variant-alternates: character-variant(zeta-2, zeta);
}
```

Obverse

ЄNO....ЧP̄RSƧЧV̄V̄SƧϤĄĢIЧPNS

En o[nom(ati) t]u p(at)r(os) (και) tu y(io)u (και) tu agiu pn(eumato)s.

ENO....UP̄RSTUŪ̄KAITṲĄG IUPNS

Reverse

LЄON|SCONSƧA|NƧIHOS..|SƧOIbAṢ.|LIṢROM|AIOH

Leon (και) Constantinos [pi]stoi bas[i]lis Romaion.

LEON|KAICONSTA|NTINOS..|STOIBAṢ.|LIṢROM|AION

Figure 27. *Byzantine seal text displayed with character variants*

EXAMPLE 24

In the figure above, the text in red is rendered using a font containing character variants that mimic the character forms found on a Byzantine seal from the 8th century A.D. Two lines below is the same text displayed in a font without variants. Note the two variants for U and N used on the seal.

```
@font-feature-values Athena Ruby {
  @character-variant {
    leo-B: 2 1;
    leo-M: 13 3;
    leo-alt-N: 14 1;
    leo-N: 14 2;
    leo-T: 20 1;
    leo-U: 21 2;
    leo-alt-U: 21 4;
  }
}

p {
  font-variant: discretionary-ligatures,
                character-variant(leo-B, leo-M, leo-N, leo-T, leo-U);
}

span.alt-N {
  font-variant-alternates: character-variant(leo-alt-N);
}

span.alt-U {
  font-variant-alternates: character-variant(leo-alt-U);
}

<p>ENO....UP̄RSTU<span class="alt-U">U</span>ŪKAITṲĄĢIUPNS</p>

<p>LEON|KAICONSTA|NTI<span class="alt-N">N</span>OS..|STOIBAṢ.|LIṢROM|AIO<span class="alt-N">N</span></p>
```

6.10 East Asian text rendering: the font-variant-east-asian property

Name:	***font-variant-east-asian***
Value:	normal | [*<east-asian-variant-values>* || *<east-asian-width-values>* || ruby]
Initial:	normal
Applies to:	all elements
Inherited:	yes
Percentages:	N/A
Media:	visual
Computed value:	as specified
Animatable:	no

Allows control of glyph substitution and sizing in East Asian text.

<east-asian-variant-values> = [jis78 | jis83 | jis90 | jis04 | simplified | traditional]

<east-asian-width-values> = [full-width | proportional-width]

Individual values have the following meanings:

normal
: None of the features listed below are enabled.

jis78
: Enables rendering of JIS78 forms (OpenType feature: `jp78`).

jis83
: Enables rendering of JIS83 forms (OpenType feature: `jp83`).

jis90
: Enables rendering of JIS90 forms (OpenType feature: `jp90`).

jis04
: Enables rendering of JIS2004 forms (OpenType feature: `jp04`).

: The various JIS variants reflect the glyph forms defined in different Japanese national standards. Fonts generally include glyphs defined by the most recent national standard but it's sometimes necessary to use older variants, to match signage for example.

simplified
: Enables rendering of simplified forms (OpenType feature: `smpl`).

traditional
: Enables rendering of traditional forms (OpenType feature: `trad`).

The ‘simplified’ and ‘traditional’ values allow control over the glyph forms for characters which have been simplified over time but for which the older, traditional form is still used in some contexts. The exact set of characters and glyph forms will vary to some degree by context for which a given font was designed.

full-width

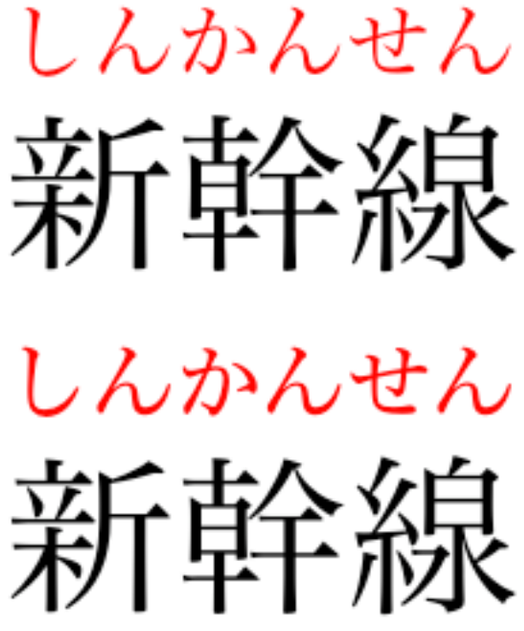
Enables rendering of full-width variants (OpenType feature: `fwid`).

proportional-width

Enables rendering of proportionally-spaced variants (OpenType feature: `pwid`).

ruby

Enables display of ruby variant glyphs (OpenType feature: `ruby`). Since ruby text is generally smaller than the associated body text, font designers can design special glyphs for use with ruby that are more readable than scaled down versions of the default glyphs. Only glyph selection is affected, there is no associated font scaling or other change that affects line layout. The red ruby text below is shown with default glyphs (top) and with ruby variant glyphs (bottom). Note the slight difference in stroke thickness.



6.11 Overall shorthand for font rendering: the font-variant property

Name:	***font-variant***
Value:	normal | none | [*<common-lig-values>* || *<discretionary-lig-values>* || *<historical-lig-values>* || *<contextual-alt-values>* || *stylistic(<feature-value-name>)* || *historical-forms* || *styleset(<feature-value-name> #)* || *character-variant(<feature-value-name> #)* || *swash(<feature-value-name>)* || *ornaments(<feature-value-name>)* || *annotation(<feature-value-name>)* || [small-caps | all-small-caps | petite-caps | all-petite-caps | unicase | titling-caps] || *<numeric-figure-values>* || *<numeric-spacing-values>* || *<numeric-fraction-values>* || ordinal || slashed-zero || *<east-asian-variant-values>* || *<east-asian-width-values>* || ruby]
Initial:	normal
Applies to:	all elements
Inherited:	yes
Percentages:	see individual properties
Media:	visual
Computed value:	see individual properties
Animatable:	see individual properties

The ‘font-variant’ property is a shorthand for all font-variant subproperties. The value ***‘normal’*** resets all subproperties of ‘font-variant’ to their inital value. The ***‘none’*** value sets ‘font-variant-ligatures’ to ‘none’ and resets all other font feature properties to their initial value. Like other shorthands, using ‘font-variant’ resets unspecified ‘font-variant’ subproperties to their initial values. It does not reset the values of either ‘font-language-override’ or ‘font-feature-settings’.

6.12 Low-level font feature settings control: the font-feature-settings property

Name:	***font-feature-settings***
Value:	normal | *<feature-tag-value>* #
Initial:	normal
Applies to:	all elements
Inherited:	yes
Percentages:	N/A
Media:	visual
Computed value:	as specified
Animatable:	no

This property provides low-level control over OpenType font features. It is intended as a way of providing access to font features that are not widely used but are needed for a particular use case.

Authors should generally use ‘font-variant’ and its related subproperties whenever possible and only use this property for special cases where its use is the only way of accessing a particular infrequently used font feature.

```
/* enable small caps and use second swash alternate */
font-feature-settings: "smcp", "swsh" 2;
```

A value of ***‘normal’*** means that no change in glyph selection or positioning occurs due to this property.

Feature tag values have the following syntax:

```
<feature-tag-value> = <string> [ <integer> | on | off ]?
```

The <string> is a case-sensitive OpenType feature tag. As specified in the OpenType specification, feature tags contain four ASCII characters. Tag strings longer or shorter than four characters, or containing characters outside the U+20–7E codepoint range are invalid. Feature tags need only match a feature tag defined in the font, so they are not limited to explicitly registered OpenType features. Fonts defining custom feature tags should follow the tag name rules defined in the OpenType specification [OPENTYPE-FEATURES].

Feature tags not present in the font are ignored; a user agent must not attempt to synthesize fallback behavior based on these feature tags. The one exception is that user agents may synthetically support the `kern` feature with fonts that contain kerning data in the form of a ‘kern’ table but lack `kern` feature support in the ‘GPOS’ table.

> In general, authors should use the ‘font-kerning’ property to explicitly enable or disable kerning since this property always affects fonts with either type of kerning data.

If present, a value indicates an index used for glyph selection. An <integer> value must be 0 or greater. A value of 0 indicates that the feature is disabled. For boolean features, a value of 1 enables the feature. For non-boolean features, a value of 1 or greater enables the feature and indicates the feature selection index. A value of ‘on’ is synonymous with 1 and ‘off’ is synonymous with 0. If the value is omitted, a value of 1 is assumed.

```
font-feature-settings: "dlig" 1;       /* dlig=1 enable discretionary ligatures */
font-feature-settings: "smcp" on;      /* smcp=1 enable small caps */
font-feature-settings: 'c2sc';         /* c2sc=1 enable caps to small caps */
font-feature-settings: "liga" off;     /* liga=0 no common ligatures */
font-feature-settings: "tnum", 'hist'; /* tnum=1, hist=1 enable tabular numbers and historical forms */
font-feature-settings: "tnum" "hist";  /* invalid, need a comma-delimited list */
font-feature-settings: "silly" off;    /* invalid, tag too long */
font-feature-settings: "PKRN";         /* PKRN=1 enable custom feature */
font-feature-settings: dlig;           /* invalid, tag must be a string */
```

When values greater than the range supported by the font are specified, the behavior is explicitly undefined. For boolean features,

in general these will enable the feature. For non-boolean features, out of range values will in general be equivalent to a 0 value. However, in both cases the exact behavior will depend upon the way the font is designed (specifically, which type of lookup is used to define the feature).

Although specifically defined for OpenType feature tags, feature tags for other modern font formats that support font features may be added in the future. Where possible, features defined for other font formats should attempt to follow the pattern of registered OpenType tags.

EXAMPLE 25

The Japanese text below will be rendered with half-width kana characters:

```
body { font-feature-settings: "hwid"; /* Half-width OpenType feature */ }

<p>毎日カレー食べてるのに、飽きない</p>
```

6.13 Font language override: the font-language-override property

Name:	***font-language-override***
Value:	normal | *<string>*
Initial:	normal
Applies to:	all elements
Inherited:	yes
Percentages:	N/A
Media:	visual
Computed value:	as specified
Animatable:	no

Normally, authors can control the use of language-specific glyph substitutions and positioning by setting the content language of an element, as described above:

```
<!-- Display text using S'gaw Karen specific features -->
<p lang="ksw">...</p>
```

In some cases, authors may need to specify a language system that differs from the content language, for example due to the need to mimic another language's typographic traditions. The ‘font-language-override’ property allows authors to explicitly specify the language system of the font, overriding the language system implied by the content language.

Values have the following meanings:

normal
: specifies that when rendering with OpenType fonts, the content language of the element is used to infer the OpenType language system

<string>
: single three-letter case-sensitive OpenType language system tag, specifies the OpenType language system to be used instead of the language system implied by the language of the element

Use of invalid OpenType language system tags must not generate a parse error but must be ignored when doing glyph selection and placement.

EXAMPLE 26

The Universal Declaration of Human Rights has been translated into a wide variety of languages. In Turkish, Article 9 of this document might be marked up as below:

```
<body lang="tr">

<h4>Madde 9</h4>
<p>Hiç kimse keyfi olarak tutuklanamaz, alıkonulanamaz veya sürülemez.</p>
```

Here the user agent uses the value of the 'lang' attribute when rendering text and appropriately renders this text without 'fi' ligatures. There is no need to use the 'font-language-override' property.

However, a given font may lack support for a specific language. In this situation authors may need to use the typographic conventions of a related language that are supported by that font:

```
<body lang="mk">     <!-- Macedonian lang code -->

body { font-language-override: "SRB"; /* Serbian OpenType language tag */ }

<h4>Член 9</h4>
<p>Никој човек нема да биде подложен на произволно апсење, притвор или прогонување.</p>
```

The Macedonian text here will be rendered using Serbian typographic conventions, with the assumption that the font specified supports Serbian.

7 Font Feature Resolution

As described in the previous section, font features can be enabled in a variety of ways, either via the use of 'font-variant' or 'font-feature-settings' in a style rule or within an `@font-face` rule. The resolution order for the union of these settings is defined below. Features defined via CSS properties are applied on top of layout engine default features.

7.1 Default features

For OpenType fonts, user agents must enable the default features defined in the OpenType documentation for a given script and writing mode. Required ligatures, common ligatures and contextual forms must be enabled by default (OpenType features: `rlig, liga, clig, calt`), along with localized forms (OpenType feature: `locl`), and features required for proper display of composed characters and marks (OpenType features: `ccmp, mark, mkmk`). These features must always be enabled, even when the value of the 'font-variant' and 'font-feature-settings' properties is 'normal'. Individual features are only disabled when explicitly overridden by the author, as when 'font-variant-ligatures' is set to 'no-common-ligatures'. For handling complex scripts such as Arabic, Mongolian or Devanagari additional features are required. For upright text within vertical text runs, vertical alternates (OpenType feature: `vert`) must be enabled.

7.2 Feature precedence

General and *font specific* font feature property settings are resolved in the order below, in ascending order of precedence. This ordering is used to construct a combined list of font features that affect a given text run.

1. Font features enabled by default, including features required for a given script.
2. If the font is defined via an `@font-face` rule, the font features implied by the font-variant descriptor in the `@font-face` rule.
3. If the font is defined via an `@font-face` rule, the font features implied by the font-feature-settings descriptor in the `@font-face` rule.
4. Font features implied by the value of the 'font-variant' property, the related 'font-variant' subproperties and any other CSS

property that uses OpenType features (e.g. the ‘font-kerning’ property).

5. Feature settings determined by properties other than ‘font-variant’ or ‘font-feature-settings’. For example, setting a non-default value for the ‘letter-spacing’ property disables common ligatures.
6. Font features implied by the value of ‘font-feature-settings’ property.

This ordering allows authors to set up a general set of defaults for fonts within their `@font-face` rules, then override them with property settings for specific elements. General property settings override the settings in `@font-face` rules and low-level font feature settings override ‘font-variant’ property settings.

For situations where the combined list of font feature settings contains more than one value for the same feature, the last value is used. When a font lacks support for a given underlying font feature, text is simply rendered as if that font feature was not enabled; font fallback does not occur and no attempt is made to synthesize the feature except where explicitly defined for specific properties.

7.3 Feature precedence examples

EXAMPLE 27

With the styles below, numbers are rendered proportionally when used within a paragraph but are shown in tabular form within tables of prices:

```
body {
  font-variant-numeric: proportional-nums;
}

table.prices td {
  font-variant-numeric: tabular-nums;
}
```

EXAMPLE 28

When the font-variant descriptor is used within an `@font-face` rule, it only applies to the font defined by that rule.

```
@font-face {
  font-family: MainText;
  src: url(http://example.com/font.woff);
  font-variant: oldstyle-nums proportional-nums styleset(1,3);
}

body {
  font-family: MainText, Helvetica;
}

table.prices td {
  font-variant-numeric: tabular-nums;
}
```

In this case, old-style numerals will be used throughout but only where the font "MainText" is used. Just as in the previous example, tabular values will be used in price tables since ‘tabular-nums’ appears in a general style rule and its use is mutually exclusive with ‘proportional-nums’. Stylistic alternate sets will only be used where MainText is used.

EXAMPLE 29

The `@font-face` rule can also be used to access font features in locally available fonts via the use of `local()` in the 'src' descriptor of the `@font-face` definition:

```
@font-face {
  font-family: BodyText;
  src: local("HiraMaruPro-W4");
  font-variant: proportional-width;
  font-feature-settings: "ital"; /* Latin italics within CJK text feature */
}

body { font-family: BodyText, serif; }
```

If available, a Japanese font "Hiragino Maru Gothic" will be used. When text rendering occurs, Japanese kana will be proportionally spaced and Latin text will be italicised. Text rendered with the fallback serif font will use default rendering properties.

EXAMPLE 30

In the example below, discretionary ligatures are enabled only for a downloadable font but are disabled within spans of class "special":

```
@font-face {
  font-family: main;
  src: url(fonts/ffmeta.woff) format("woff");
  font-variant: discretionary-ligatures;
}

body         { font-family: main, Helvetica; }
span.special { font-variant-ligatures: no-discretionary-ligatures; }
```

Suppose one adds a rule using 'font-feature-settings' to enable discretionary ligatures:

```
body         { font-family: main, Helvetica; }
span         { font-feature-settings: "dlig"; }
span.special { font-variant-ligatures: no-discretionary-ligatures; }
```

In this case, discretionary ligatures *will* be rendered within spans of class "special". This is because both the 'font-feature-settings' and 'font-variant-ligatures' properties apply to these spans. Although the 'no-discretionary ligatures' setting of 'font-variant-ligatures' effectively disables the OpenType `dlig` feature, because the 'font-feature-settings' is resolved after that, the 'dlig' value reenables discretionary ligatures.

8 Object Model

The contents of `@font-face` and `@font-feature-values` rules can be accessed via the following extensions to the CSS Object Model.

8.1 The `CSSFontFaceRule` interface

The ***CSSFontFaceRule*** interface represents a `@font-face` rule.

```
interface CSSFontFaceRule : CSSRule {
  attribute DOMString family;
  attribute DOMString src;
  attribute DOMString style;
  attribute DOMString weight;
  attribute DOMString stretch;
  attribute DOMString unicodeRange;
  attribute DOMString variant;
  attribute DOMString featureSettings;
}
```

The DOM Level 2 Style specification [DOM-LEVEL-2-STYLE] defined a different variant of this rule. This definition supercedes that one.

8.2 The `CSSFontFeatureValuesRule` interface

The `CSSRule` interface is extended as follows:

```
partial interface CSSRule {
  const unsigned short FONT_FEATURE_VALUES_RULE = 14;
}
```

The ***CSSFontFeatureValuesRule*** interface represents a `@font-feature-values` rule.

```
interface CSSFontFeatureValuesRule : CSSRule {
  attribute DOMString fontFamily;
  readonly attribute CSSFontFeatureValuesMap annotation;
  readonly attribute CSSFontFeatureValuesMap ornaments;
  readonly attribute CSSFontFeatureValuesMap stylistic;
  readonly attribute CSSFontFeatureValuesMap swash;
  readonly attribute CSSFontFeatureValuesMap characterVariant;
  readonly attribute CSSFontFeatureValuesMap styleset;
}

[MapClass(DOMString, sequence<unsigned long>)]
interface CSSFontFeatureValuesMap {
  void set(DOMString featureValueName,
           (unsigned long or sequence<unsigned long>) values);
}
```

fontFamily **of type `DOMString`**

The list of one or more font families for which a given set of feature values is defined.

value maps of type `CSSFontFeatureValuesMap`, readonly

Maps of feature values associated with feature value names for a given ‘font-variant-alternates’ value type

Each value map attribute of `CSSFontFeatureValuesRule` reflects the values defined via a corresponding feature value block. Thus, the *annotation* attribute contains the values contained within a `@annotation` feature value block, the *ornaments* attribute contains the values contained with a `@ornaments` feature value block and so forth.

The `CSSFontFeatureValuesMap` interface uses the default map class methods but the `set` method has different behavior. It takes a sequence of unsigned integers and associates it with a given `featureValueName`. The method behaves the same as the default map class method except that a single unsigned long value is treated as a sequence of a single value. The method throws an exception if an invalid number of values is passed in. If the associated feature value block only allows a limited number of values, the `set` method throws an `InvalidAccessError` exception when the input sequence to `set` contains more than the limited number of values. See the description of multi-valued feature value definitions for details on the maximum number of values allowed for a given type of feature value block. The `get` method always returns a sequence of values, even if the sequence only contains a single value.

Appendix A: Mapping platform font properties to CSS properties

This appendix is included as background for some of the problems and situations that are described in other sections. It should be viewed as informative only.

Font properties in CSS are designed to be independent of the underlying font formats used; they can be used to specify bitmap fonts, Type1 fonts, SVG fonts in addition to the common TrueType and OpenType fonts. But there are facets of the TrueType and OpenType formats that often cause confusion for authors and present challenges to implementers on different platforms.

Originally developed at Apple, TrueType was designed as an outline font format for both screen and print. Microsoft joined Apple in developing the TrueType format and both platforms have supported TrueType fonts since then. Font data in the TrueType format consists of a set of tables distinguished with common four-letter tag names, each containing a specific type of data. For example, naming information, including copyright and license information, is stored in the 'name' table. The *character map* ('cmap') table contains a mapping of character encodings to glyphs. Apple later added additional tables for supporting enhanced typographic functionality; these are now called Apple Advanced Typography, or AAT, fonts. Microsoft and Adobe developed a separate set of tables for advanced typography and called their format OpenType [OPENTYPE].

In many cases the font data used under Microsoft Windows or Linux is slightly different from the data used under Apple's Mac OS X because the TrueType format allowed for explicit variation across platforms. This includes font metrics, names and *character map* data.

Specifically, font family name data is handled differently across platforms. For TrueType and OpenType fonts these names are contained in the 'name' table, in name records with name ID 1. Mulitple names can be stored for different locales, but Microsoft recommends fonts always include at least a US English version of the name. On Windows, Microsoft made the decision for backwards compatibility to limit this family name to a maximum of four faces; for larger groupings the "preferred family" (name ID 16) or "WWS family" (name ID 21) can be used. Other platforms such as OSX don't have this limitation, so the family name is used to define all possible groupings.

Other name table data provides names used to uniquely identify a specific face within a family. The full font name (name ID 4) and the Postscript name (name ID 6) describe a single face uniquely. For example, the bold face of the Gill Sans family has a fullname of "Gill Sans Bold" and a Postscript name of "GillSans-Bold". There can be multiple localized versions of the fullname for a given face, but the Postscript name is always a unique name made from a limited set of ASCII characters.

On various platforms, different names are used to search for a font. For example, with the Windows GDI CreateIndirectFont API, either a family or fullname can be used to lookup a face, while on Mac OS X the CTFontCreateWithName API call is used to lookup a given face using the fullname and Postscript name. Under Linux, the fontconfig API allows fonts to be searched using any of these names. In situations where platform API's automatically substitute other font choices, it may be necessary to verify a returned font matches a given name.

The weight of a given face can be determined via the usWeightClass field of the OS/2 table or inferred from the style name (name ID 2). Likewise, the width can be determined via the usWidthClass of the OS/2 table or inferred from the style name. For historical reasons related to synthetic bolding at weights 200 or lower with the Windows GDI API, font designers have sometimes skewed values in the OS/2 table to avoid these weights.

Rendering complex scripts that use contextual shaping such as Thai, Arabic and Devanagari requires features present only in OpenType or AAT fonts. Currently, complex script rendering is supported on Windows and Linux using OpenType font features while both OpenType and AAT font features are used under Mac OS X.

Changes

Changes from the July 2013 CSS3 Fonts Last Call Working Draft

- reorder feature precedence such that features implied by other CSS properties override 'font-variant' settings
- switched examples to use .woff files
- revised wording of font fetching algorithm
- drop 'auto' value for 'font-size-adjust'
- clarify effect of 'font-size-adjust' on 'line-height'
- allow user agents to synthesize OpenType `kern` feature
- add example of ordinals and associated markup

- minor editorial cleanups

Acknowledgments

I'd like to thank Tal Leming, Jonathan Kew and Christopher Slye for all their help and feedback. John Hudson was kind enough to take the time to explain the subtleties of OpenType language tags and provided the example of character variant usage for displaying text on Byzantine seals. Ken Lunde and Eric Muller provided valuable feedback on CJK OpenType features and Unicode variation selectors. The idea for supporting font features by using 'font-variant' subproperties originated with Håkon Wium Lie, Adam Twardoch and Tal Leming. Elika Etemad supplied some of the initial design ideas for the `@font-feature-values` rule. Thanks also to House Industries for allowing the use of Ed Interlock in the discretionary ligatures example.

A special thanks to Robert Bringhurst for the sublime mind expansion that is *The Elements of Typographic Style*.

Conformance

Document Conventions

Conformance requirements are expressed with a combination of descriptive assertions and RFC 2119 terminology. The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in the normative parts of this document are to be interpreted as described in RFC 2119. However, for readability, these words do not appear in all uppercase letters in this specification.

All of the text of this specification is normative except sections explicitly marked as non-normative, examples, and notes. [RFC2119]

Examples in this specification are introduced with the words "for example" or are set apart from the normative text with `class="example"`, like this:

> EXAMPLE 31
>
> This is an example of an informative example.

Informative notes begin with the word "Note" and are set apart from the normative text with `class="note"`, like this:

> Note, this is an informative note.

Conformance Classes

Conformance to CSS Fonts Level 3 Module is defined for three conformance classes:

style sheet
: A CSS style sheet.

renderer
: A UA that interprets the semantics of a style sheet and renders documents that use them.

authoring tool
: A UA that writes a style sheet.

A style sheet is conformant to CSS Fonts Level 3 Module if all of its declarations that use properties defined in this module have values that are valid according to the generic CSS grammar and the individual grammars of each property as given in this module.

A renderer is conformant to CSS Fonts Level 3 Module if, in addition to interpreting the style sheet as defined by the appropriate specifications, it supports all the features defined by CSS Fonts Level 3 Module by parsing them correctly and rendering the document accordingly. However, the inability of a UA to correctly render a document due to limitations of the device does not make

the UA non-conformant. (For example, a UA is not required to render color on a monochrome monitor.)

An authoring tool is conformant to CSS Fonts Level 3 Module if it writes style sheets that are syntactically correct according to the generic CSS grammar and the individual grammars of each feature in this module, and meet all other conformance requirements of style sheets as described in this module.

Partial Implementations

So that authors can exploit the forward-compatible parsing rules to assign fallback values, CSS renderers **must** treat as invalid (and ignore as appropriate) any at-rules, properties, property values, keywords, and other syntactic constructs for which they have no usable level of support. In particular, user agents **must not** selectively ignore unsupported component values and honor supported values in a single multi-value property declaration: if any value is considered invalid (as unsupported values must be), CSS requires that the entire declaration be ignored.

Experimental Implementations

To avoid clashes with future CSS features, the CSS2.1 specification reserves a prefixed syntax for proprietary and experimental extensions to CSS.

Prior to a specification reaching the Candidate Recommendation stage in the W3C process, all implementations of a CSS feature are considered experimental. The CSS Working Group recommends that implementations use a vendor-prefixed syntax for such features, including those in W3C Working Drafts. This avoids incompatibilities with future changes in the draft.

Non-Experimental Implementations

Once a specification reaches the Candidate Recommendation stage, non-experimental implementations are possible, and implementors should release an unprefixed implementation of any CR-level feature they can demonstrate to be correctly implemented according to spec.

To establish and maintain the interoperability of CSS across implementations, the CSS Working Group requests that non-experimental CSS renderers submit an implementation report (and, if necessary, the testcases used for that implementation report) to the W3C before releasing an unprefixed implementation of any CSS features. Testcases submitted to W3C are subject to review and correction by the CSS Working Group.

Further information on submitting testcases and implementation reports can be found from on the CSS Working Group's website at http://www.w3.org/Style/CSS/Test/. Questions should be directed to the public-css-testsuite@w3.org mailing list.

CR Exit Criteria

Remove this section unless/until the module is in CR.

For this specification to be advanced to Proposed Recommendation, there must be at least two independent, interoperable implementations of each feature. Each feature may be implemented by a different set of products, there is no requirement that all features be implemented by a single product. For the purposes of this criterion, we define the following terms:

independent
: each implementation must be developed by a different party and cannot share, reuse, or derive from code used by another qualifying implementation. Sections of code that have no bearing on the implementation of this specification are exempt from this requirement.

interoperable
: passing the respective test case(s) in the official CSS test suite, or, if the implementation is not a Web browser, an equivalent test. Every relevant test in the test suite should have an equivalent test created if such a user agent (UA) is to be used to claim interoperability. In addition if such a UA is to be used to claim interoperability, then there must one or more additional UAs which can also pass those equivalent tests in the same way for the purpose of interoperability. The equivalent tests must be made publicly available for the purposes of peer review.

implementation

a user agent which: **(1)** implements the specification. **(2)** is available to the general public. The implementation may be a shipping product or other publicly available version (i.e., beta version, preview release, or “nightly build”). Non-shipping product releases must have implemented the feature(s) for a period of at least one month in order to demonstrate stability. **(3)** is not experimental (i.e., a version specifically designed to pass the test suite and is not intended for normal usage going forward).

The specification will remain Candidate Recommendation for at least six months.

References

Normative References

[CHARMOD]

Martin J. Dürst; et al. *Character Model for the World Wide Web 1.0: Fundamentals.* 15 February 2005. W3C Recommendation. URL: http://www.w3.org/TR/2005/REC-charmod-20050215/

[CORS]

Anne van Kesteren. *Cross-Origin Resource Sharing.* 29 January 2013. W3C Candidate Recommendation. (Work in progress.) URL: http://www.w3.org/TR/2013/CR-cors-20130129/

[CSS21]

Bert Bos; et al. *Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification.* 7 June 2011. W3C Recommendation. URL: http://www.w3.org/TR/2011/REC-CSS2-20110607/

[CSS3VAL]

Håkon Wium Lie; Tab Atkins; Elika J. Etemad. *CSS Values and Units Module Level 3.* 30 July 2013. W3C Candidate Recommendation. (Work in progress.) URL: http://www.w3.org/TR/2013/CR-css3-values-20130730/

[HTML5]

Robin Berjon; et al. *HTML5.* 6 August 2013. W3C Candidate Recommendation. (Work in progress.) URL: http://www.w3.org/TR/2013/CR-html5-20130806/

[OPEN-FONT-FORMAT]

Information technology — Coding of audio-visual objects — Part 22: Open Font Format. International Organization for Standardization. ISO/IEC 14496-22:2009. URL: http://standards.iso.org/ittf/PubliclyAvailableStandards/c052136_ISO_IEC_14496-22_2009%28E%29.zip

[OPENTYPE]

OpenType specification. Microsoft. URL: http://www.microsoft.com/typography/otspec/default.htm

[OPENTYPE-FEATURES]

OpenType feature registry. Microsoft. URL: http://www.microsoft.com/typography/otspec/featurelist.htm

[RFC2119]

S. Bradner. *Key words for use in RFCs to Indicate Requirement Levels.* Internet RFC 2119. URL: http://www.ietf.org/rfc/rfc2119.txt

[UAX15]

Mark Davis; Ken Whistler. *Unicode Normalization Forms.* 31 August 2012. Unicode Standard Annex #15. URL: http://www.unicode.org/reports/tr15/

[UAX29]

Mark Davis. *Unicode Text Segmentation.* 12 September 2012. Unicode Standard Annex #29. URL: http://www.unicode.org/reports/tr29/

[UNICODE]

The Unicode Consortium. *The Unicode Standard.* 2012. Defined by: The Unicode Standard, Version 6.2.0 (Mountain View, CA: The Unicode Consortium, 2012. ISBN 978-1-936213-07-8), as updated from time to time by the publication of new versions URL: http://www.unicode.org/standard/versions/enumeratedversions.html

Other References

[AAT-FEATURES]

Apple Advanced Typography font feature registry. Apple. URL: http://developer.apple.com/fonts/registry/

[ARABIC-TYPO]

Huda Smitshuijzen AbiFares. *Arabic Typography: A Comprehensive Sourcebook.* Saqi Books. 2001. ISBN 0-86356-347-3.

[CHARMOD-NORM]

François Yergeau; et al. *Character Model for the World Wide Web 1.0: Normalization.* 1 May 2012. W3C Working Draft. (Work in progress.) URL: http://www.w3.org/TR/2012/WD-charmod-norm-20120501/

[CJKV-INFO-PROCESSING]

Ken Lunde. *CJKV Information Processing, Second Edition.* O'Reilly Media, Inc. 2009. ISBN 0-596-51447-1.

[CSS3-CONDITIONAL]

L. David Baron. *CSS Conditional Rules Module Level 3.* 4 April 2013. W3C Candidate Recommendation. (Work in progress.) URL: http://www.w3.org/TR/2013/CR-css3-conditional-20130404/

[CSS3TEXT]

Elika J. Etemad; Koji Ishii. *CSS Text Module Level 3.* 13 November 2012. W3C Working Draft. (Work in progress.) URL: http://www.w3.org/TR/2012/WD-css3-text-20121113/

[DIGITAL-TYPOGRAPHY]

Richard Rubinstein. *Digital Typography, An Introduction to Type and Composition for Computer System Design.* Addison-Wesley. 1988. ISBN 0-201-17633-5.

[DOM-LEVEL-2-STYLE]

Chris Wilson; Philippe Le Hégaret; Vidur Apparao. *Document Object Model (DOM) Level 2 Style Specification.* 13 November 2000. W3C Recommendation. URL: http://www.w3.org/TR/2000/REC-DOM-Level-2-Style-20001113/

[ELEMTYPO]

Robert Bringhurst. *The Elements of Typographic Style, Version 4.* Hartley & Marks. 2013. ISBN 0-88179-212-8.

[LANGCULTTYPE]

John D. Berry, Ed. *Language Culture Type.* Graphis. 2001. ISBN 1-932026-01-0.

[OPENTYPE-FONT-GUIDE]

OpenType User Guide. FontShop International. URL: https://www.fontfont.com/staticcontent/downloads/FF_OT_User_Guide.pdf

[RASTER-TRAGEDY]

Beat Stamm. *The Raster Tragedy at Low-Resolution Revisited.* 7 December 2011. URL: http://www.rastertragedy.com/

[WINDOWS-GLYPH-PROC]

John Hudson. *Windows Glyph Processing.* Microsoft Typogrraphy. URL: http://www.microsoft.com/typography/developers/opentype/default.htm

Index

Property index

Property	Values	Initial	Applies to	Inh.	Percentages	Media
font	[[<'font-style'> || <font-variant-css21> || <'font-weight'> || <'font-stretch']? <'font-size'> [/ <'line-height'>]? <'font-family'>] | caption | icon | menu | message-box | small-caption | status-bar	see individual properties	all elements	yes	see individual properties	visual
font-family	[<family-name> | <generic-family>] #	depends on user agent	all elements	yes	N/A	visual
font-feature-settings	normal | <feature-tag-value> #	normal	all elements	yes	N/A	visual
font-kerning	auto | normal | none	auto	all elements	yes	N/A	visual
font-language-override	normal | <string>	normal	all elements	yes	N/A	visual
font-size	<absolute-size> | <relative-size> | <length> | <percentage>	medium	all elements	yes	refer to parent element's font size	visual
font-size-adjust	none | <number>	none	all elements	yes	N/A	visual
font-stretch	normal | ultra-condensed | extra-condensed | condensed | semi-condensed | semi-expanded | expanded | extra-expanded | ultra-expanded	normal	all elements	yes	N/A	visual
font-style	normal | italic | oblique	normal	all elements	yes	N/A	visual
font-synthesis	none | [weight || style]	weight style	all elements	yes	N/A	visual
	normal | none | [<common-lig-values> || <discretionary-lig-values> ||					

font-variant	<historical-lig-values> || <contextual-alt-values> || stylistic(<feature-value-name>) || historical-forms || styleset(<feature-value-name> #) || character-variant(<feature-value-name> #) || swash(<feature-value-name>) || ornaments(<feature-value-name>) || annotation(<feature-value-name>) || [small-caps	all-small-caps	petite-caps	all-petite-caps	unicase	titling-caps] || <numeric-figure-values> || <numeric-spacing-values> || <numeric-fraction-values> || ordinal || slashed-zero || <east-asian-variant-values> || <east-asian-width-values> || ruby]	normal	all elements	yes	see individual properties	visual
font-variant-alternates	normal	[stylistic(<feature-value-name>)		historical-forms		styleset(<feature-value-name> #)		character-variant(<feature-value-name> #)		swash(<feature-value-name>)		ornaments(<feature-value-name>)		annotation(<feature-value-name>)]	normal	all elements	yes	N/A	visual
font-variant-caps	normal	small-caps	all-small-caps	petite-caps	all-petite-caps	unicase	titling-caps	normal	all elements	yes	N/A	visual
font-variant-east-asian	normal	[<east-asian-variant-values>		<east-asian-width-values>		ruby]	normal	all elements	yes	N/A	visual
font-variant-ligatures	normal	none	[<common-lig-values>		<discretionary-lig-values>		<historical-lig-values>		<contextual-alt-values>]	normal	all elements	yes	N/A	visual
font-variant-numeric	normal	[<numeric-figure-values>		<numeric-spacing-values>		<numeric-fraction-values>		ordinal		slashed-zero]	normal	all elements	yes	N/A	visual
font-variant-position	normal	sub	super	normal	all elements	yes	N/A	visual
font-weight	normal	bold	bolder	lighter	100	200	300	400	500	600	700	800	900	normal	all elements	yes	N/A	visual

Descriptor	Value	Initial	Percentages	Media
font-family	<family-name>	N/A		
font-feature-settings	normal	<feature-tag-value> #	normal		
font-stretch	normal	ultra-condensed	extra-condensed	condensed	semi-condensed	semi-expanded	expanded	extra-expanded	ultra-expanded	normal		
font-style	normal	italic	oblique	normal		
font-variant	normal	none	[<common-lig-values> || <discretionary-lig-values> || <historical-lig-values> || <contextual-alt-values> || stylistic(<feature-value-name>) || historical-forms || styleset(<feature-value-name> #) || character-variant(<feature-value-name> #) || swash(<feature-value-name>) || ornaments(<feature-value-name>) || annotation(<feature-value-name>) || [small-caps	all-small-caps	petite-caps	all-petite-caps	unicase	titling-caps] || <numeric-figure-values> || <numeric-spacing-values> || <numeric-fraction-values> || ordinal || slashed-zero || <east-asian-variant-values> || <east-asian-width-values> || ruby]	normal		
font-weight	normal	bold	100	200	300	400	500	600	700	800	900	normal		
src	[<url> [format(<string> #)]?	<font-face-name>] #	N/A		
unicode-range	<urange> #	U+0-10FFFF		