5 Selectors

Contents

Note: Several sections of this specification have been updated by other specifications. Please, see "Cascading Style Sheets (CSS) — The Official Definition" in the latest *CSS Snapshot* for a list of specifications and the sections they replace.
The CSS Working Group is also developing CSS level 2 revision 2 (CSS 2.2).

5.1 Pattern matching

In CSS, pattern matching rules determine which style rules apply to elements in the document tree. These patterns, called selectors, may range from simple element names to rich contextual patterns. If all conditions in the pattern are true for a certain element, the selector *matches* the element.

The case-sensitivity of document language element names in selectors depends on the document language. For example, in HTML, element names are case-insensitive, but in XML they are case-sensitive.

The following table summarizes CSS 2.1 selector syntax:

Pattern	Meaning	Described in section
*	Matches any element.	Universal selector

E	Matches any E element (i.e., an element of type E).	Type selectors
E F	Matches any F element that is a descendant of an E element.	Descendant selectors
E > F	Matches any F element that is a child of an element E.	Child selectors
E:first-child	Matches element E when E is the first child of its parent.	The :first-child pseudo-class
E:link E:visited	Matches element E if E is the source anchor of a hyperlink of which the target is not yet visited (:link) or already visited (:visited).	The link pseudo-classes
E:active E:hover E:focus	Matches E during certain user actions.	The dynamic pseudo-classes
E:lang(c)	Matches element of type E if it is in (human) language c (the document language specifies how language is determined).	The :lang() pseudo-class
E + F	Matches any F element immediately preceded by a sibling element E.	Adjacent selectors
E[foo]	Matches any E element with the "foo" attribute set (whatever the value).	Attribute selectors
E[foo="warning"]	Matches any E element whose "foo" attribute value is exactly equal to "warning".	Attribute selectors
E [foo~="warning"]	Matches any E element whose "foo" attribute value is a list of space-separated values, one of which is exactly equal to "warning".	Attribute selectors
E[lang|="en"]	Matches any E element whose "lang" attribute has a hyphen-separated list of values beginning (from the left) with "en".	Attribute selectors
DIV.warning	*Language specific.* (In HTML, the same as DIV[class~="warning"].)	Class selectors
E#myid	Matches any E element with ID equal to "myid".	ID selectors

5.2 Selector syntax

A *simple selector* is either a type selector or universal selector followed immediately by zero or more attribute selectors, ID selectors, or pseudo-classes, in any order. The simple selector matches if all of its components match.

Note: the terminology used here in CSS 2.1 is different from what is used in CSS3. For example, a "simple selector" refers to a smaller part of

a selector in CSS3 than in CSS 2.1. See the CSS3 Selectors module [CSS3SEL].

A *selector* is a chain of one or more simple selectors separated by combinators. *Combinators* are: white space, ">", and "+". White space may appear between a combinator and the simple selectors around it.

The elements of the document tree that match a selector are called *subjects* of the selector. A selector consisting of a single simple selector matches any element satisfying its requirements. Prepending a simple selector and combinator to a chain imposes additional matching constraints, so the subjects of a selector are always a subset of the elements matching the last simple selector.

One pseudo-element may be appended to the last simple selector in a chain, in which case the style information applies to a subpart of each subject.

5.2.1 Grouping

When several selectors share the same declarations, they may be grouped into a comma-separated list.

In this example, we condense three rules with identical declarations into one. Thus,

```
h1 { font-family: sans-serif }
h2 { font-family: sans-serif }
h3 { font-family: sans-serif }
```

is equivalent to:

```
h1, h2, h3 { font-family: sans-serif }
```

CSS offers other "shorthand" mechanisms as well, including multiple declarations and shorthand properties.

5.3 Universal selector

The universal selector, written "*", matches the name of any element type. It matches any single element in the document tree.

If the universal selector is not the only component of a simple selector, the "*" may be omitted. For example:

- `*[lang=fr]` and `[lang=fr]` are equivalent.
- `*.warning` and `.warning` are equivalent.
- `*#myid` and `#myid` are equivalent.

5.4 Type selectors

A *type selector* matches the name of a document language element type. A type selector matches every instance of the element type in the document tree.

The following rule matches all H1 elements in the document tree:

```
h1 { font-family: sans-serif }
```

5.5 Descendant selectors

At times, authors may want selectors to match an element that is the descendant of another element in the document tree (e.g., "Match those EM elements that are contained by an H1 element"). Descendant selectors express such a relationship in a pattern. A descendant selector is made up of two or more selectors separated by white space. A descendant selector of the form "A B" matches when an element B is an arbitrary descendant of some ancestor element A.

For example, consider the following rules:

```
h1 { color: red }
em { color: red }
```

Although the intention of these rules is to add emphasis to text by changing its color, the effect will be lost in a case such as:

```
<H1>This headline is <EM>very</EM> important</H1>
```

We address this case by supplementing the previous rules with a rule that sets the text color to blue whenever an EM occurs anywhere within an H1:

```
h1 { color: red }
em { color: red }
h1 em { color: blue }
```

The third rule will match the EM in the following fragment:

```
<H1>This <SPAN class="myclass">headline
is <EM>very</EM> important</SPAN></H1>
```

The following selector:

```
div * p
```

matches a P element that is a grandchild or later descendant of a DIV element. Note the white space on either side of the "*" is not part of the universal selector; the white space is a combinator indicating that the DIV must be the ancestor of some element, and that that element must be an ancestor of the P.

The selector in the following rule, which combines descendant and attribute selectors, matches any element that (1) has the "href" attribute set and (2) is inside a P that is itself inside a DIV:

```
div p *[href]
```

5.6 Child selectors

A *child selector* matches when an element is the child of some element. A child selector is made up of two or more selectors separated by ">".

The following rule sets the style of all P elements that are children of BODY:

```
body > P { line-height: 1.3 }
```

The following example combines descendant selectors and child selectors:

```
div ol>li p
```

It matches a P element that is a descendant of an LI; the LI element must be the child of an OL element; the OL element must be a descendant of a DIV. Notice that the optional white space around the ">" combinator has been left out.

For information on selecting the first child of an element, please see the section on the :first-child pseudo-class below.

5.7 Adjacent sibling selectors

Adjacent sibling selectors have the following syntax: E1 + E2, where E2 is the subject of the selector. The selector matches if E1 and E2 share the same parent in the document tree and E1 immediately precedes E2, ignoring non-element nodes (such as text nodes and comments).

Thus, the following rule states that when a P element immediately follows a MATH element, it should not be indented:

```
math + p { text-indent: 0 }
```

The next example reduces the vertical space separating an H1 and an H2 that immediately follows it:

```
h1 + h2 { margin-top: -5mm }
```

The following rule is similar to the one in the previous example, except that it adds a class selector. Thus, special formatting only occurs when H1 has `class="opener"`:

```
h1.opener + h2 { margin-top: -5mm }
```

5.8 Attribute selectors

CSS 2.1 allows authors to specify rules that match elements which have certain attributes defined in the source document.

5.8.1 Matching attributes and attribute values

Attribute selectors may match in four ways:

[att]
: Match when the element sets the "att" attribute, whatever the value of the attribute.

[att=val]
: Match when the element's "att" attribute value is exactly "val".

[att~=val]
: Represents an element with the `att` attribute whose value is a white space-separated list of words, one of which is exactly "val". If "val" contains white space, it will never represent anything (since the words are *separated* by spaces). If "val" is the empty string, it will never represent anything either.

[att|=val]

Represents an element with the `att` attribute, its value either being exactly "val" or beginning with "val" immediately followed by "-" (U +002D). This is primarily intended to allow language subcode matches (e.g., the `hreflang` attribute on the a element in HTML) as described in BCP 47 ([BCP47]) or its successor. For `lang` (or `xml:lang`) language subcode matching, please see the `:lang` pseudo-class.

Attribute values must be identifiers or strings. The case-sensitivity of attribute names and values in selectors depends on the document language.

For example, the following attribute selector matches all H1 elements that specify the "title" attribute, whatever its value:

```
h1[title] { color: blue; }
```

In the following example, the selector matches all SPAN elements whose "class" attribute has exactly the value "example":

```
span[class=example] { color: blue; }
```

Multiple attribute selectors can be used to refer to several attributes of an element, or even several times to the same attribute.

Here, the selector matches all SPAN elements whose "hello" attribute has exactly the value "Cleveland" and whose "goodbye" attribute has exactly the value "Columbus":

```
span[hello="Cleveland"][goodbye="Columbus"] { color: blue; }
```

The following selectors illustrate the differences between "=" and "~=". The first selector will match, for example, the value "copyright copyleft copyeditor" for the "rel" attribute. The second selector will only match when the "href" attribute has the value "http://www.w3.org/".

```
a[rel~="copyright"]
a[href="http://www.w3.org/"]
```

The following rule hides all elements for which the value of the "lang" attribute is "fr" (i.e., the language is French).

```
*[lang=fr] { display : none }
```

The following rule will match for values of the "lang" attribute that begin with "en", including "en", "en-US", and "en-cockney":

```
*[lang|="en"] { color : red }
```

Similarly, the following aural style sheet rules allow a script to be read aloud in different voices for each role:

```
DIALOGUE[character=romeo]
     { voice-family: "Laurence Olivier", charles, male }

DIALOGUE[character=juliet]
     { voice-family: "Vivien Leigh", victoria, female }
```

5.8.2 Default attribute values in DTDs

Matching takes place on attribute values in the document tree. Default attribute values may be defined in a DTD or elsewhere, but cannot always be selected by attribute selectors. Style sheets should be designed so that they work even if the default values are not included in the document tree.

More precisely, a UA may, but is *not* required to, read an "external subset" of the DTD but *is* required to look for default attribute values in the document's "internal subset." (See [XML10] for definitions of these subsets.) Depending on the UA, a default attribute value defined in the external subset of the DTD might or might not appear in the document tree.

A UA that recognizes an XML namespace [XMLNAMESPACES] may, but is not required to, use its knowledge of that namespace to treat default attribute values as if they were present in the document. (E.g., an XHTML UA is not required to use its built-in knowledge of the XHTML DTD.)

Note that, typically, implementations choose to ignore external subsets.

For example, consider an element EXAMPLE with an attribute "notation" that has a default value of "decimal". The DTD fragment might be

```
<!ATTLIST EXAMPLE notation (decimal,octal) "decimal">
```

If the style sheet contains the rules

```
EXAMPLE[notation=decimal] { /*... default property settings ...*/ }
EXAMPLE[notation=octal]   { /*... other settings...*/ }
```

the first rule might not match elements whose "notation" attribute is set by default, i.e., not set explicitly. To catch all cases, the attribute selector for the default value must be dropped:

```
EXAMPLE                   { /*... default property settings ...*/ }
EXAMPLE[notation=octal]   { /*... other settings...*/ }
```

Here, because the selector `EXAMPLE[notation=octal]` is more specific than the type selector alone, the style declarations in the second rule will override those in the first for elements that have a "notation" attribute value of "octal". Care has to be taken that all property declarations that are to apply only to the default case are overridden in the non-default cases' style rules.

5.8.3 Class selectors

Working with HTML, authors may use the period (`.`) notation as an alternative to the ~= notation when representing the `class` attribute. Thus, for HTML, `div.value` and `div[class~=value]` have the same meaning. The attribute value must immediately follow the "period" (`.`). UAs may apply selectors using the period (.) notation in XML documents if the UA has namespace specific knowledge that allows it to determine which attribute is the "class" attribute for the respective namespace. One such example of namespace specific knowledge is the prose in the specification for a particular namespace (e.g., SVG 1.1 [SVG11] describes the SVG "class" attribute and

how a UA should interpret it, and similarly MathML 3.0 [MATH30] describes the MathML "class" attribute.)

For example, we can assign style information to all elements with `class~="pastoral"` as follows:

```
*.pastoral { color: green }  /* all elements with class~=pastoral */
```

or just

```
.pastoral { color: green }  /* all elements with class~=pastoral */
```

The following assigns style only to H1 elements with `class~="pastoral"`:

```
H1.pastoral { color: green }  /* H1 elements with class~=pastoral */
```

Given these rules, the first H1 instance below would not have green text, while the second would:

```
<H1>Not green</H1>
<H1 class="pastoral">Very green</H1>
```

To match a subset of "class" values, each value must be preceded by a ".".

For example, the following rule matches any P element whose "class" attribute has been assigned a list of space-separated values that includes "pastoral" and "marine":

```
p.marine.pastoral { color: green }
```

This rule matches when `class="pastoral blue aqua marine"` but does not match for `class="pastoral blue"`.

Note. *CSS gives so much power to the "class" attribute, that authors could conceivably design their own "document language" based on elements with almost no associated presentation (such as DIV and SPAN in HTML) and assigning style information through the "class" attribute. Authors should avoid this practice since the structural elements of a document language often have recognized and accepted meanings and author-defined classes may not.*

Note: *If an element has multiple class attributes, their values must be concatenated with spaces between the values before searching for the class. As of this time the working group is not aware of any manner in which this situation can be reached, however, so this behavior is explicitly non-normative in this specification.*

5.9 ID selectors

Document languages may contain attributes that are declared to be of type ID. What makes attributes of type ID special is that no two such attributes can have the same value; whatever the document language, an ID attribute can be used to uniquely identify its element. In HTML all ID attributes are named "id"; XML applications may name ID attributes differently, but the same restriction applies.

The ID attribute of a document language allows authors to assign an identifier to one element instance in the document tree. CSS ID selectors match an element instance based on its identifier. A CSS ID selector contains a "#" immediately followed by the ID value, which must be an identifier.

Note that CSS does not specify how a UA knows the ID attribute of an element. The UA may, e.g., read a document's DTD, have the information hard-coded or ask the user.

The following ID selector matches the H1 element whose ID attribute has the value "chapter1":

```
h1#chapter1 { text-align: center }
```

In the following example, the style rule matches the element that has the ID value "z98y". The rule will thus match for the P element:

```
<HEAD>
  <TITLE>Match P</TITLE>
  <STYLE type="text/css">
    *#z98y { letter-spacing: 0.3em }
  </STYLE>
</HEAD>
<BODY>
   <P id=z98y>Wide text</P>
</BODY>
```

In the next example, however, the style rule will only match an H1 element that has an ID value of "z98y". The rule will not match the P element in this example:

```
<HEAD>
  <TITLE>Match H1 only</TITLE>
  <STYLE type="text/css">
    H1#z98y { letter-spacing: 0.5em }
  </STYLE>
</HEAD>
<BODY>
   <P id=z98y>Wide text</P>
</BODY>
```

ID selectors have a higher specificity than attribute selectors. For example, in HTML, the selector #p123 is more specific than `[id=p123]` in terms of the cascade.

Note. *In XML 1.0 [XML10], the information about which attribute contains an element's IDs is contained in a DTD. When parsing XML, UAs do not always read the DTD, and thus may not know what the ID of an element is. If a style sheet designer knows or suspects that this will be the case, he should use normal attribute selectors instead: `[name=p371]` instead of #p371. However, the cascading order of normal attribute selectors is different from ID selectors. It may be necessary to add an "! important" priority to the declarations: `[name=p371] {color: red ! important}`.*

If an element has multiple ID attributes, all of them must be treated as IDs for that element for the purposes of the ID selector. Such a situation could be reached using mixtures of xml:id [XMLID], DOM3 Core [DOM-LEVEL-3-CORE], XML DTDs [XML10] and namespace-specific knowledge.

5.10 Pseudo-elements and pseudo-classes

In CSS 2.1, style is normally attached to an element based on its position in the document tree. This simple model is sufficient for many cases, but some common publishing scenarios may not be possible due to the structure of the document tree. For instance, in HTML 4 (see [HTML4]), no element refers to the first line of a paragraph, and therefore no simple CSS selector may refer to it.

CSS introduces the concepts of *pseudo-elements* and *pseudo-classes* to permit formatting based on information that lies outside the document tree.

- Pseudo-elements create abstractions about the document tree beyond those specified by the document language. For instance, document languages do not offer mechanisms to access the first letter or first line of an element's content. CSS pseudo-elements allow style sheet designers to refer to this otherwise inaccessible information. Pseudo-elements may also provide style sheet designers a way to assign style to content that does not exist in the source document (e.g., the :before and :after pseudo-elements give access to generated content).
- Pseudo-classes classify elements on characteristics other than their name, attributes or content; in principle characteristics that cannot be deduced from the document tree. Pseudo-classes may be dynamic, in the sense that an element may acquire or lose a pseudo-class while a user interacts with the document. The exceptions are ':first-child', which *can* be deduced from the document tree, and ':lang()', which can be deduced from the document tree in some cases.

Neither pseudo-elements nor pseudo-classes appear in the document source or document tree.

Pseudo-classes are allowed anywhere in selectors while pseudo-elements may only be appended after the last simple selector of the selector.

Pseudo-element and pseudo-class names are case-insensitive.

Some pseudo-classes are mutually exclusive, while others can be applied simultaneously to the same element. In case of conflicting rules, the normal cascading order determines the outcome.

5.11 Pseudo-classes

5.11.1 :first-child pseudo-class

The :first-child pseudo-class matches an element that is the first child element of some other element.

In the following example, the selector matches any P element that is the first child of a DIV element. The rule suppresses indentation for the first paragraph of a DIV:

```
div > p:first-child { text-indent: 0 }
```

This selector would match the P inside the DIV of the following fragment:

```
<P> The last P before the note.
<DIV class="note">
   <P> The first P inside the note.
</DIV>
```

but would not match the second P in the following fragment:

```
<P> The last P before the note.
<DIV class="note">
   <H2>Note</H2>
   <P> The first P inside the note.
</DIV>
```

The following rule sets the font weight to 'bold' for any EM element that is some descendant of a P element that is a first child:

```
p:first-child em { font-weight : bold }
```

Note that since anonymous boxes are not part of the document tree, they are not counted when calculating the first child.

For example, the EM in:

```
<P>abc <EM>default</EM>
```

is the first child of the P.

The following two selectors are equivalent:

```
* > a:first-child   /* A is first child of any element */
a:first-child       /* Same */
```

5.11.2 The link pseudo-classes: :link and :visited

User agents commonly display unvisited links differently from previously visited ones. CSS provides the pseudo-classes ':link' and ':visited' to distinguish them:

- The :link pseudo-class applies for links that have not yet been visited.
- The :visited pseudo-class applies once the link has been visited by the user.

UAs may return a visited link to the (unvisited) ':link' state at some point.

The two states are mutually exclusive.

The document language determines which elements are hyperlink source anchors. For example, in HTML4, the link pseudo-classes apply to A elements with an "href" attribute. Thus, the following two CSS 2.1 declarations have similar effect:

```
a:link { color: red }
:link  { color: red }
```

If the following link:

```
<A class="external" href="http://out.side/">external link</A>
```

has been visited, this rule:

```
a.external:visited { color: blue }
```

will cause it to be blue.

Note. It is possible for style sheet authors to abuse the :link and :visited pseudo-classes to determine which sites a user has visited without the user's consent.

UAs may therefore treat all links as unvisited links, or implement other measures to preserve the user's privacy while rendering visited and unvisited links differently. See [P3P] for more information about handling privacy.

5.11.3 The dynamic pseudo-classes: :hover, :active, and :focus

Interactive user agents sometimes change the rendering in response to user actions. CSS provides three pseudo-classes for common cases:

- The :hover pseudo-class applies while the user designates an element (with some pointing device), but does not activate it. For example, a visual user agent could apply this pseudo-class when the cursor (mouse pointer) hovers over a box generated by the element. User agents not supporting interactive media do not have to support this pseudo-class. Some conforming user agents supporting interactive media may not be able to support this pseudo-class (e.g., a pen device).
- The :active pseudo-class applies while an element is being activated by the user. For example, between the times the user presses the mouse button and releases it.
- The :focus pseudo-class applies while an element has the focus (accepts keyboard events or other forms of text input).

An element may match several pseudo-classes at the same time.

CSS does not define which elements may be in the above states, or how the states are entered and left. Scripting may change whether elements react to user events or not, and different devices and UAs may have different ways of pointing to, or activating elements.

CSS 2.1 does not define if the parent of an element that is ':active' or ':hover' is also in that state.

User agents are not required to reflow a currently displayed document due to pseudo-class transitions. For instance, a style sheet may specify that the 'font-size' of an :active link should be larger than that of an inactive link, but since this may cause letters to change position when the reader selects the link, a UA may ignore the corresponding style rule.

```
a:link    { color: red }    /* unvisited links */
a:visited { color: blue }   /* visited links   */
a:hover   { color: yellow } /* user hovers     */
a:active  { color: lime }   /* active links    */
```

Note that the A:hover must be placed after the A:link and A:visited rules, since otherwise the cascading rules will hide the 'color' property of the A:hover rule. Similarly, because A:active is placed after A:hover, the active

color (lime) will apply when the user both activates and hovers over the A element.

An example of combining dynamic pseudo-classes:

```
a:focus { background: yellow }
a:focus:hover { background: white }
```

The last selector matches A elements that are in pseudo-class :focus and in pseudo-class :hover.

For information about the presentation of focus outlines, please consult the section on dynamic focus outlines.

Note. *In CSS1, the ':active' pseudo-class was mutually exclusive with ':link' and ':visited'. That is no longer the case. An element can be both ':visited' and ':active' (or ':link' and ':active') and the normal cascading rules determine which style declarations apply.*

Note. *Also note that in CSS1, the ':active' pseudo-class only applied to links.*

5.11.4 The language pseudo-class: :lang

If the document language specifies how the human language of an element is determined, it is possible to write selectors in CSS that match an element based on its language. For example, in HTML [HTML4], the language is determined by a combination of the "lang" attribute, the META element, and possibly by information from the protocol (such as HTTP headers). XML uses an attribute called xml:lang, and there may be other document language-specific methods for determining the language.

The pseudo-class ':lang(C)' matches if the element is in language C. Whether there is a match is based solely on the identifier C being either equal to, or a hyphen-separated substring of, the element's language value, in the same way as if performed by the '|=' operator. The matching of C against the element's language value is performed case-insensitively for characters within the ASCII range. The identifier C does not have to be a valid language name.

C must not be empty.

Note: It is recommended that documents and protocols indicate language using codes from BCP 47 [BCP47] or its successor, and by means of "xml:lang" attributes in the case of XML-based documents [XML10]. See "FAQ: Two-letter or three-letter language codes."

The following rules set the quotation marks for an HTML document that is either in Canadian French or German:

```
html:lang(fr-ca) { quotes: '« ' ' »' }
html:lang(de) { quotes: '»' '«' '\2039' '\203A' }
:lang(fr) > Q { quotes: '« ' ' »' }
:lang(de) > Q { quotes: '»' '«' '\2039' '\203A' }
```

The second pair of rules actually set the 'quotes' property on Q elements according to the language of its parent. This is done because the choice of quote marks is typically based on the language of the element around the

quote, not the quote itself: like this piece of French “à l'improviste” in the middle of an English text uses the English quotation marks.

Note the difference between [lang|=xx] and :lang(xx). In this HTML example, only the BODY matches [lang|=fr] (because it has a LANG attribute) but both the BODY and the P match :lang(fr) (because both are in French).

```
<body lang=fr>
  <p>Je suis Français.</p>
</body>
```

5.12 Pseudo-elements

Pseudo-elements behave just like real elements in CSS with the exceptions described below and elsewhere.

Note that the sections below do not define the exact rendering of ':first-line' and ':first-letter' in all cases. A future level of CSS may define them more precisely.

5.12.1 The :first-line pseudo-element

The :first-line pseudo-element applies special styles to the contents of the first formatted line of a paragraph. For instance:

```
p:first-line { text-transform: uppercase }
```

The above rule means "change the letters of the first line of every paragraph to uppercase". However, the selector "P:first-line" does not match any real HTML element. It does match a pseudo-element that conforming user agents will insert at the beginning of every paragraph.

Note that the length of the first line depends on a number of factors, including the width of the page, the font size, etc. Thus, an ordinary HTML paragraph such as:

```
<P>This is a somewhat long HTML
paragraph that will be broken into several
lines. The first line will be identified
by a fictional tag sequence. The other lines
will be treated as ordinary lines in the
paragraph.</P>
```

the lines of which happen to be broken as follows:

```
THIS IS A SOMEWHAT LONG HTML PARAGRAPH THAT
will be broken into several lines. The first
line will be identified by a fictional tag
sequence. The other lines will be treated as
ordinary lines in the paragraph.
```

might be "rewritten" by user agents to include the *fictional tag sequence* for :first-line. This fictional tag sequence helps to show how properties are inherited.

```
<P><P:first-line> This is a somewhat long HTML
paragraph that </P:first-line> will be broken into several
lines. The first line will be identified
by a fictional tag sequence. The other lines
will be treated as ordinary lines in the
paragraph.</P>
```

If a pseudo-element breaks up a real element, the desired effect can often be described by a fictional tag sequence that closes and then re-opens the element. Thus, if we mark up the previous paragraph with a SPAN element:

```
<P><SPAN class="test"> This is a somewhat long HTML
paragraph that will be broken into several
lines.</SPAN> The first line will be identified
by a fictional tag sequence. The other lines
will be treated as ordinary lines in the
paragraph.</P>
```

the user agent could simulate start and end tags for SPAN when inserting the fictional tag sequence for :first-line.

```
<P><P:first-line><SPAN class="test"> This is a
somewhat long HTML
paragraph that will </SPAN></P:first-line><SPAN class="test"> be
broken into several
lines.</SPAN> The first line will be identified
by a fictional tag sequence. The other lines
will be treated as ordinary lines in the
paragraph.</P>
```

The :first-line pseudo-element can only be attached to a block container element.

The "first formatted line" of an element may occur inside a block-level descendant in the same flow (i.e., a block-level descendant that is not positioned and not a float). E.g., the first line of the DIV in `<DIV><P>This line...</P></DIV>` is the first line of the P (assuming that both P and DIV are block-level).

The first line of a table-cell or inline-block cannot be the first formatted line of an ancestor element. Thus, in `<DIV><P STYLE="display: inline-block">Hello<BR>Goodbye</P> etcetera</DIV>` the first formatted line of the DIV is not the line "Hello".

> Note that the first line of the P in this fragment: `<p><br>First...` does not contain any letters (assuming the default style for BR in HTML 4). The word "First" is not on the first formatted line.

A UA should act as if the fictional start tags of the first-line pseudo-elements were nested just inside the innermost enclosing block-level element. (Since CSS1 and CSS2 were silent on this case, authors should not rely on this behavior.) Here is an example. The fictional tag sequence for

```
<DIV>
  <P>First paragraph</P>
  <P>Second paragraph</P>
</DIV>
```

is

```
<DIV>
  <P><DIV:first-line><P:first-line>First paragraph</P:first-line></DIV:first-line
  <P><P:first-line>Second paragraph</P:first-line></P>
</DIV>
```

The :first-line pseudo-element is similar to an inline-level element, but with certain restrictions. The following properties apply to a :first-line pseudo-element: font properties, color property, background properties, 'word-spacing', 'letter-spacing', 'text-decoration', 'text-transform', and 'line-height'. UAs may apply other properties as well.

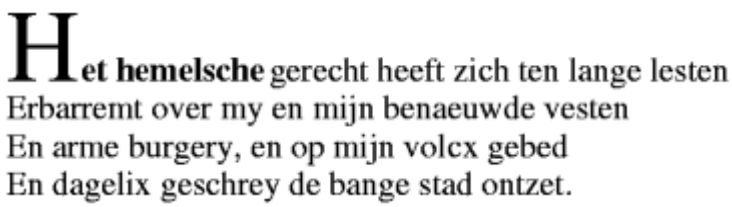
5.12.2 The :first-letter pseudo-element

The :first-letter pseudo-element must select the first letter of the first line of a block, if it is not preceded by any other content (such as images or inline tables) on its line. The :first-letter pseudo-element may be used for "initial caps" and "drop caps", which are common typographical effects. This type of initial letter is similar to an inline-level element if its 'float' property is 'none', otherwise it is similar to a floated element.

These are the properties that apply to :first-letter pseudo-elements: font properties, 'text-decoration', 'text-transform', 'letter-spacing', 'word-spacing' (when appropriate), 'line-height', 'float', 'vertical-align' (only if 'float' is 'none'), margin properties, padding properties, border properties, color property, background properties. UAs may apply other properties as well. To allow UAs to render a typographically correct drop cap or initial cap, the UA may choose a line-height, width and height based on the shape of the letter, unlike for normal elements. CSS3 is expected to have specific properties that apply to first-letter.

This example shows a possible rendering of an initial cap. Note that the 'line-height' that is inherited by the first-letter pseudo-element is 1.1, but the UA in this example has computed the height of the first letter differently, so that it does not cause any unnecessary space between the first two lines. Also note that the fictional start tag of the first letter is inside the SPAN, and thus the font weight of the first letter is normal, not bold as the SPAN:

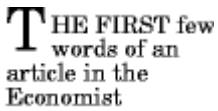
```
p { line-height: 1.1 }
p:first-letter { font-size: 3em; font-weight: normal }
span { font-weight: bold }
...
<p><span>Het hemelsche</span> gerecht heeft zich ten lange lesten<br>
Erbarremt over my en mijn benaeuwde vesten<br>
En arme burgery, en op mijn volcx gebed<br>
En dagelix geschrey de bange stad ontzet.
```



The following CSS 2.1 will make a drop cap initial letter span about two lines:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<HTML>
 <HEAD>
  <TITLE>Drop cap initial letter</TITLE>
  <STYLE type="text/css">
   P              { font-size: 12pt; line-height: 1.2 }
   P:first-letter { font-size: 200%; font-style: italic;
                    font-weight: bold; float: left }
   SPAN           { text-transform: uppercase }
  </STYLE>
 </HEAD>
 <BODY>
  <P><SPAN>The first</SPAN> few words of an article
    in The Economist.</P>
 </BODY>
</HTML>
```

This example might be formatted as follows:



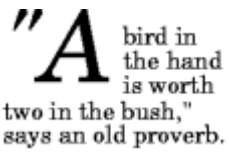
The fictional tag sequence is:

```
<P>
<SPAN>
<P:first-letter>
T
</P:first-letter>he first
</SPAN>
few words of an article in the Economist.
</P>
```

Note that the :first-letter pseudo-element tags abut the content (i.e., the initial character), while the :first-line pseudo-element start tag is inserted right after the start tag of the block element.

In order to achieve traditional drop caps formatting, user agents may approximate font sizes, for example to align baselines. Also, the glyph outline may be taken into account when formatting.

Punctuation (i.e, characters defined in Unicode [UNICODE] in the "open" (Ps), "close" (Pe), "initial" (Pi). "final" (Pf) and "other" (Po) punctuation classes), that precedes or follows the first letter should be included, as in:



The ':first-letter' also applies if the first letter is in fact a digit, e.g., the "6" in "67 million dollars is a lot of money."

The :first-letter pseudo-element applies to block container elements.

The :first-letter pseudo-element can be used with all such elements that contain text, or that have a descendant in the same flow that contains text. A UA should act as if the fictional start tag of the first-letter pseudo-element is just before the first text of the element, even if that first text is in a descendant.

Here is an example. The fictional tag sequence for this HTML fragment:

```
<div>
<p>The first text.
```

is:

```
<div>
<p><div:first-letter><p:first-letter>T</...></...>he first text.
```

The first letter of a table-cell or inline-block cannot be the first letter of an ancestor element. Thus, in <DIV><P STYLE="display: inline-block">Hello
Goodbye</P> etcetera</DIV> the first letter of the DIV is not the letter "H". In fact, the DIV does not have a first letter.

The first letter must occur on the first formatted line. For example, in this fragment: <p>
First... the first line does not contain any letters and ':first-letter' does not match anything (assuming the default style for BR in HTML 4). In particular, it does not match the "F" of "First."

If an element is a list item ('display: list-item'), the ':first-letter' applies to the first letter in the principal box after the marker. UAs may ignore ':first-letter' on list items with 'list-style-position: inside'. If an element has ':before' or ':after' content, the ':first-letter applies to the first letter of the element *including* that content.

E.g., after the rule 'p:before {content: "Note: "}', the selector 'p:first-letter' matches the "N" of "Note".

Some languages may have specific rules about how to treat certain letter combinations. In Dutch, for example, if the letter combination "ij" appears at the beginning of a word, both letters should be considered within the :first-letter pseudo-element.

If the letters that would form the first-letter are not in the same element, such as "'T" in <p>'T..., the UA may create a first-letter pseudo-element from one of the elements, both elements, or simply not create a pseudo-element.

Similarly, if the first letter(s) of the block are not at the start of the line (for example due to bidirectional reordering), then the UA need not create the pseudo-element(s).

The following example illustrates how overlapping pseudo-elements may interact. The first letter of each P element will be green with a font size of '24pt'. The rest of the first formatted line will be 'blue' while the rest of the paragraph will be 'red'.

```
p { color: red; font-size: 12pt }
p:first-letter { color: green; font-size: 200% }
p:first-line { color: blue }

<P>Some text that ends up on two lines</P>
```

Assuming that a line break will occur before the word "ends", the fictional tag sequence for this fragment might be:

```
<P>
<P:first-line>
<P:first-letter>
S
</P:first-letter>ome text that
</P:first-line>
ends up on two lines
</P>
```

Note that the :first-letter element is inside the :first-line element. Properties set on :first-line are inherited by :first-letter, but are overridden if the same property is set on :first-letter.

5.12.3 The :before and :after pseudo-elements

The ':before' and ':after' pseudo-elements can be used to insert generated content before or after an element's content. They are explained in the section on generated text.

```
h1:before {content: counter(chapno, upper-roman) ". "}
```

When the :first-letter and :first-line pseudo-elements are applied to an element having content generated using :before and :after, they apply to the first letter or line of the element including the generated content.

```
p.special:before {content: "Special! "}
p.special:first-letter {color: #ffd800}
```

This will render the "S" of "Special!" in gold.