

# **XMLmind XML Editor - Support of Cascading Style Sheets (W3C CSS)**

**Hussein Shafie  
Pixware**

`<xmleditor-support+xmlmind.com>`

---

# XMLmind XML Editor - Support of Cascading Style Sheets (W3C CSS)

Hussein Shafie

Pixware

<xmleditor-support@xmlmind.com>

Publication date October 28, 2010

## Abstract

This document describes the subset of CSS2 supported by XXE, as well as advanced ``proprietary extensions" needed to style complex XML documents.

---

I. Guide .....	1
1. Introduction .....	2
2. Restrictions .....	3
3. Extensions related to generated content .....	6
1. Replaced content .....	6
2. Generated content .....	6
4. Other extensions .....	8
1. Built-in CSS rules .....	8
2. CSS3 selectors .....	9
3. Styling an element which contains a specific processing instruction .....	9
4. Styling an element which contains a specific child element .....	9
5. Specifying namespaces .....	10
6. Inserting in generated content the name of the element which is the target of the CSS rule .....	11
7. Dynamic evaluation of property values .....	12
7.1. Simple dynamic evaluation of property values .....	12
7.2. Using custom code to extend the CSS style sheet .....	12
8. New values for the display property .....	13
9. Rendering repeating elements as a table .....	14
10. Making a table look like a spreadsheet .....	15
11. Collapsible blocks and tables .....	16
12. Styling comments and processing instructions .....	18
13. Styling element attributes .....	19
14. :property() extension pseudo class .....	22
15. url() is XML catalog aware .....	23
16. Modularizing a complex CSS style sheet using @property-group and @property-value .....	23
16.1. @property-group .....	23
16.2. @property-value .....	25
17. marker-offset: fill .....	28
18. If needed, selectors can use default attribute values .....	29
19. Simple, fast, purely declarative counters .....	29
II. Reference .....	31
5. Content objects .....	32
1. add-attribute-button .....	33
2. attributes .....	33
3. check-box .....	34
4. collapser .....	35
5. combo-box .....	35
6. command-button .....	36
7. component .....	37
8. convert-button .....	38
9. date-field .....	38
10. date-time-picker .....	39
11. date-picker .....	41
12. delete-button .....	41
13. drag-source .....	41
14. drop-site .....	42
15. file-name-field .....	42
16. gadget .....	44
17. icon .....	44
18. indicator .....	45
19. insert-after-button .....	46
20. insert-before-button .....	46
21. insert-button .....	46
22. insert-same-after-button .....	47
23. insert-same-before-button .....	47
24. image .....	47
25. image-viewport .....	48
26. label .....	51

---

27. list .....	51
28. number-field .....	52
29. radio-buttons .....	53
30. remove-attribute-button .....	54
31. replace-button .....	55
32. set-attribute-button .....	55
33. text .....	55
34. text-area .....	56
35. text-field .....	56
36. time-picker .....	57
37. value-editor .....	57
38. xpath .....	57
6. Content layouts .....	59
1. division .....	59
2. paragraph .....	59
3. rows .....	59
7. Display values supported for generated content .....	61
1. display: inline .....	61
2. display: block .....	62
3. display: list-item .....	67
4. display: table .....	67
5. display: table-row-group .....	68
6. display: table-row .....	70
7. display: table-cell .....	71

---

# Part I. Guide

---

---

# Chapter 1. Introduction

XMLmind XML Editor (XXE for short) supports a subset of CSS2 and a few CSS3 features.

The role of the CSS style sheet in XXE is to make the XML document easy to read (get rid of the tree view, no visible tags, etc) and to make its structure (chapter, section, list, list item, etc) easy to understand.

This is very different from the role of CSS style sheets in Web browsers, for which the CSS standard has been designed.

In practice, this means:

- You really need to design a CSS style sheet *specifically for XML authoring*. For that, no need to be WYSIWYG, that is,
  - you should not try to emulate what will be displayed in the browser after the conversion of the XML document to HTML;
  - you should not try to emulate what will be displayed in Acrobat™ Reader after conversion of the XML document to PDF.

Note that XXE supports enough CSS to make your XML documents look WYSIRN (**W**hat **Y**ou **S**ee **I**s **R**eally **N**eat).

- Unless you are styling XML data (or a mix of XML document/XML data) rather than XML documents, you should restrict yourself from using XMLmind proprietary extensions. You can style 99% of any type of XML document using the subset of CSS2 supported by XXE. (The remaining 1% is solved by the `image()` [47] or the `image-viewport()` [48] content objects.)

---

# Chapter 2. Restrictions

## Important

The properties not listed in the following two tables are *not supported by XXE*.

The following properties can be inherited whether explicitly (**inherit** keyword) or implicitly (inherited property).

For all properties except line-height where the specified number is inherited (which is the correct behavior), the inherited value is the actual value not the computed value.

Property	Value	Restrictions
background-color	<i>color</i>   transparent   inherit   normal <sup>a</sup>	-
border	<i>width</i> [ <i>style color</i> ?]?   inherit	Order is strictly width, style, color
border-color	<i>side_value</i> {1,4}	-
border-bottom-color	<i>color</i>   transparent   inherit	-
border-left-color	"	-
border-right-color	"	-
border-top-color	"	-
border-top-left-radius	<i>length</i>   <i>percentage</i> [ <i>length</i>   <i>percentage</i> ]?   inherit	<b>Note</b>  All the following <i>border-radius</i> properties are CSS3, and not CSS2, properties.
border-top-right-radius	"	
border-bottom-left-radius	"	
border-bottom-right-radius	"	
border-radius	[ <i>length</i>   <i>percentage</i> ]{1,4} [ / [ <i>length</i>   <i>percentage</i> ]{1,4} ]?	-
border-style	none   dotted   dashed   solid   double   groove   ridge   inset   outset	No hidden
border-width	thin   thick   medium   <i>length</i>   inherit	-
color	<i>color</i>   inherit   normal <sup>a</sup>	-
counter-reset, counter-increment	[ <i>identifier integer</i> ? ]+   none   inherit	-
display	none   inline   block   list-item   marker   table   inline-table   table-row-group   table-header-group   table-footer-group   table-row   table-column-group   table-column   table-cell   table-caption   inline-block   tree   inherit	No run-in, compact.
font	[ <i>style weight</i> ?]? <i>size family</i>   inherit	Order is strictly style then weight
font-family	[[ <i>name</i>   <i>generic</i> ],]* [ <i>name</i>   <i>generic</i> ]   inherit   normal <sup>a</sup>	The generic font families <i>cursive</i> and <i>fantasy</i> are not supported.

Property	Value	Restrictions
font-size	medium   small   large   x-small   x-large   xx-small   xx-large   smaller   larger   <i>length</i>   <i>percentage</i>   inherit	-
font-style	normal   italic   oblique   inherit	italic and oblique are aliases
font-weight	normal   bold   inherit	No <i>N00</i> , bolder, lighter
line-height	normal   <i>number</i>   inherit	No <i>length</i> , <i>percentage</i>
list-style-image	<i>URI</i>   none     inherit	Also supports icon( <i>name</i> ).
list-style-position	outside   inside   inherit	-
list-style-type	decimal   lower-alpha   upper-alpha   lower-roman   upper-roman   none   inherit	No decimal-leading-zero, hebrew, armenian, lower-greek, etc.
list-style	<i>type</i> [ <i>position</i> [ <i>image</i> ]? ]?   inherit	Order is strictly type then position then image.
margin	<i>side_value</i> {1,4}	-
margin-bottom	<i>length</i>   auto   inherit	No <i>percentage</i>
margin-left	"	-
margin-right	"	-
margin-top	"	-
padding	<i>side_value</i> {1,4}	-
padding-bottom	<i>length</i>   inherit	No <i>percentage</i>
padding-left	"	-
padding-right	"	-
padding-top	"	-
text-align	left   right   center   inherit	No justify
text-decoration	none   underline   overline   line-through   inherit	No blink
text-indent	<i>length</i>   inherit	No <i>percentage</i>
vertical-align	baseline   middle   sub   super   text-top   top   text-bottom   bottom   inherit	No <i>length</i> , <i>percentage</i>
white-space	normal   pre   nowrap   inherit	-

<sup>a</sup>"normal" is a non-standard value meaning: same value as the one specified in the user's preferences.

The following properties cannot be inherited whether explicitly (**inherit** keyword) or implicitly (inherited property).

Property	Value	Restrictions
border-spacing	<i>length length?</i>	-
caption-side	top   bottom	left, right, inherit are not supported.
content	<i>string</i>   <i>uri</i>   attr( <i>X</i> )   open-quote   close-quote   no-open-quote   no-close-quote   counter( <i>name</i> )   counter( <i>name</i> , <i>style</i> )   counters( <i>name</i> , <i>separ</i> )   counters( <i>name</i> , <i>separ</i> , <i>style</i> )   disc   circle   square   see extensions	No-open-quote, no-close-quote are ignored.  Counter styles are limited to: decimal, lower-alpha, lower-latin, upper-alpha, upper-latin, lower-roman, upper-roman.
height	<i>length</i>   auto	No <i>percentage</i> . This property is currently ignored.



Property	Value	Restrictions
marker-offset	<i>length</i>   auto   fill	No <i>percentage</i> .
width	<i>length</i>   auto	No <i>percentage</i> . This property is currently only useful to specify the minimum width of a table cell.

Other restrictions:

- The CSS box decorations (**border**, **padding**, etc) are not supported for inlined elements. The **background-color** is the only property supported for such elements.

A workaround consists in using `display: inline-block; white-space: nowrap;`. See Section 8, “New values for the display property” [13]. The `white-space: nowrap;` is generally needed because, by default, “inline-blocks” and “inline-tables” are made as narrow as possible.

- Inserting block elements inside inlined elements is not supported. It will not crash the XML editor, but the result will be ugly. However inserting element having property `display: inline-block;` or property `display: inline-table;` inside inlined elements should work fine.
- The border properties, except **border-color**, cannot be specified individually for each side of the box.
- When a CSS box has at least one rounded corner (specified using any of the **border-radius** properties), the only supported **border-styles** are: `dotted`, `dashed` and `solid`.
- **:first-letter** and **:first-line** pseudo-elements are ignored.
- The **!important** specifier is ignored.

---

# Chapter 3. Extensions related to generated content

## Tip

Rules which use extensions specific to XMLmind XML Editor may be specified in `@media XMLmind-XML-Editor` constructs (identifier `XMLmind-XML-Editor` being case-insensitive). Example:

```
@media XMLmind-XML-Editor {  
  img {  
    content: image(attr(src));  
  }  
}
```

Elaborate examples of generated content can be found in `XXE_install_dir/demo/bugreport/bugreport.css` and in `XXE_install_dir/addon/config/common/css/xmldata.css`.

## 1. Replaced content

XXE not only supports generated content but also supports *replaced content*. This means that **content** may be used for any element and not only for **:before** and **:after** pseudo-elements. When used for an actual element, it replaces what is normally displayed for this element.

Therefore, in what follows, generated content generally means generated or replaced content.

## 2. Generated content

Extensions related to generated content fall in three categories:

- Content objects [32].

Standard CSS only supports text and images. Example: `content: url(images/right.png) "foo=" attr(foo);`. XXE supports other ways of specifying text and images as well as using controls (buttons, comboboxes, etc) as generated content.

Example:

```
img {  
  content: image(attr(src));  
}
```

- Content layouts [59].

Standard CSS does not allow to structure and layout generated content. XXE allows for example to structure and layout generated content as an embedded table.

Example:

```
orderedProducts:before {  
  display: table-row;  
  content: row(cell("QUANTITY"),  
               cell("REFERENCE"),  
               cell("DESIGNATION"),  
               cell(content("PRICE\A", attr(currency))),  
               font-weight, bold,  
               color, white,  
               background-color, #0000A0,  
               border-width, 1,  
               border-style, solid,
```

```
}    border-top-color, gray,  
    border-bottom-color, gray,  
    border-right-color, gray,  
    border-left-color, gray);
```

- Display values supported for generated content [61].

Standard CSS only supports inline, block, marker as the value of the display property of generated content, and generated content is limited to inline and block elements. XXE does much more than this.

Example: table-row in the above example.

---

# Chapter 4. Other extensions

## Tip

Rules which use extensions specific to XMLmind XML Editor may be specified in `@media XMLmind-XML-Editor` constructs (identifier `XMLmind-XML-Editor` being case-insensitive). Example:

```
@media XMLmind-XML-Editor {  
  img {  
    content: image(attr(src));  
  }  
}
```

## 1. Built-in CSS rules

XMLmind XML Editor has built-in CSS rules mainly used to style comments and processing instructions. These built-in rules are always implicitly loaded before the rules found in a CSS file. However, nothing prevents you from overriding any of the following built-in rules.

```
*::comment,  
*::processing-instruction {  
  display: block;  
  margin: 2px;  
  white-space: pre;  
  text-align: left;  
  font-family: monospace;  
  font-style: normal;  
  font-weight: normal;  
  font-size: small;  
}  
  
*::comment {  
  background-color: #FFFFCC;  
  color: #808000;  
}  
  
*::processing-instruction {  
  background-color: #CCFFCC;  
  color: #008000;  
}  
  
*::processing-instruction(xe-formula) {  
  content: gadget("com.xmlmind.xmleditapp.spreadsheet.Formula");  
  display: inline;  
}  
  
*::read-only {  
  background-color: #E0F0F0;  
}  
  
@namespace xi url(http://www.w3.org/2001/XInclude);  
  
xi|include,  
xi\:include {  
  display: tree;  
}  
  
@media print {  
  *::comment,  
  *::processing-instruction,  
  *::processing-instruction(xe-formula) {  
    display: none;  
  }  
  
  *::read-only {
```

```
background-color: transparent;
}
}
```

In practice, this just means that you have nothing special to do to style comments, processing instructions and spreadsheet formulas (processing instruction `xxe-formula`).

## 2. CSS3 selectors

In addition to all CSS2 selectors, XXE also supports the following CSS3 selectors:

Pattern	Meaning
<code>E:last-child</code>	an <code>E</code> element, last child of its parent
<code>E:first-of-type</code>	an <code>E</code> element, first sibling of its type
<code>E:last-of-type</code>	an <code>E</code> element, last sibling of its type
<code>E:root</code>	an <code>E</code> element which is the root element of a document
<code>E:empty</code>	an <code>E</code> element which does not contain child nodes of any type
<code>[att^=val]</code>	the <code>att</code> attribute whose value begins with the prefix " <code>val</code> "
<code>[att\$=val]</code>	the <code>att</code> attribute whose value ends with the suffix " <code>val</code> "
<code>[att*=val]</code>	the <code>att</code> attribute whose value contains at least one instance of the substring " <code>val</code> "

## 3. Styling an element which contains a specific processing instruction

Use pseudo-class `:contains-processing-instruction(target)` where *target*, a CSS identifier or string, is the target of the processing instructions.

Example: display all XHTML `span`s containing one or more spreadsheet formulas with a yellow background.

```
span:contains-processing-instruction(xxe-formula) {
    background-color: yellow;
}
```

## 4. Styling an element which contains a specific child element

Use pseudo-class `:contains-element(element_name)` where *element\_name*, a CSS identifier, string or qualified name, specifies the name of child element.

Note that:

```
p:contains-element(i) {
    color: red;
}
```

is very different from:

```
p > i {
    color: red;
}
```

In the first case, the target of the CSS rule, that is the element which is styled, is `p`. In the second case, it is `i`.

Examples:

```
/* No namespace declaration before this. */

p:contains-element(i) {1
    color: red;
}

p:contains-element(|b) {2
    color: green;
}

@namespace foo "http://foo.com";

p:contains-element(foo|h1) {3
    color: blue;
}

@namespace "http://bar.com";

p:contains-element(h1) {4
    color: yellow;
}

*|*:contains-element(*|h1) {5
    text-decoration: underline;
}

*|h1 {
    display: inline;
}
```

- 1** Element with local name `p`, whatever is its namespace, containing a `i` whatever is its namespace, gets a red color.
- 2** Element with local name `p`, whatever is its namespace, containing a `{ }b`, gets a green color.
- 3** Element with local name `p`, whatever is its namespace, containing a `{http://foo.com}h1`, gets a blue color.
- 4** Element `{http://bar.com}p`, containing a `{http://bar.com}h1`, gets a yellow color.
- 5** Any element having a child with local name `h1`, whatever is the namespace of this `h1`, is to be underlined.

## 5. Specifying namespaces

Namespace support in CSS3 style sheets is outlined in Selectors. In summary:

- `@namespace` rule declares a namespace prefix and associates it to the namespace URI. Examples:

```
@namespace url(http://www.xmlmind.com/xmleditor/schema/configuration);

@namespace html url(http://www.w3.org/1999/xhtml);
```

Rule #1 specifies that element names (in selectors) without an explicit namespace component belong to the "http://www.xmlmind.com/xmleditor/schema/configuration" namespace.

Rule #2 specifies that element or attribute names with a "html" prefix belong to the "http://www.w3.org/1999/xhtml" namespace.

- Notation for qualified names is `prefix|local_name`, where character `|` is used to separate the two parts of the qualified name.

Example of element names:

```
@namespace ns url(http://www.ns.com);

ns|para { font-size: 8pt; }
ns|*    { font-size: 9pt; }
|para   { font-size: 10pt; }
```

```
*|para { font-size: 11pt; }  
para   { font-size: 11pt; }
```

**Rule #1**

will match only `para` elements in the "`http://www.ns.com`" namespace.

**Rule #2**

will match all elements in the "`http://www.ns.com`" namespace.

**Rule #3**

will match only `para` elements without any declared namespace.

**Rule #4**

will match `para` elements in any namespace (including those without any declared namespace).

**Rule #5**

is equivalent to the rule #4 because no default namespace has been defined.

Examples of attribute names:

```
@namespace ns "http://www.ns.com";  
  
[ns|role=minor] { font-size: 8pt; }  
[*|role]        { font-size: 9pt; }  
[|role]         { font-size: 10pt; }  
[role]          { font-size: 10pt; }
```

**Rule #1**

will match only elements with the attribute `role` in the "`http://www.ns.com`" namespace with the value "`minor`".

**Rule #2**

will match only elements with the attribute `role` regardless of the namespace of the attribute (including no declared namespace).

**Rule #3 and #4**

will match only elements with the attribute `role` where the attribute is not declared to be in a namespace.

Note that default namespaces do not apply to attributes.

- The `attr()` pseudo-function also supports namespaces.

```
@namespace ns "http://www.ns.com";  
  
para:before { content: attr(ns|role); }
```

The generated content inserted before "`para`" elements is the content of attribute `role` declared in the "`http://www.ns.com`" namespace.

## 6. Inserting in generated content the name of the element which is the target of the CSS rule

Standard pseudo-function `attr()` can be used to insert in generated content the value of an attribute of the element which is the target of CSS rule.

Example:

```
xref {  
  content: "xref=" attr(linkend) " ";  
}
```

Pseudo functions `element-name()`, `element-local-name()`, `element-namespace-uri()`, `element-label()` are similar to `attr()` except that they insert strings related to the name of the element which is the target of CSS rule.

Example:

```
xref {
  content: element-name() "=" attr(linkend) " ";
}
```

Pseudo-function	Description	Example
<code>element-name()</code>	The fully qualified name of the element.	<code>ns:myElement-1</code>
<code>element-local-name()</code>	Local part of element name.	<code>myElement-1</code>
<code>element-namespace-uri()</code>	Namespace URI of element name.	<code>http://acme.com/ns/foo/bar</code>
<code>element-label()</code>	Local part of element name, made more readable.	<code>My element 1</code>

## 7. Dynamic evaluation of property values

### 7.1. Simple dynamic evaluation of property values

**concatenate**(*value*, ..., *value*) may be used to specify a dynamically evaluated property value anywhere a static property value is allowed.

A dynamic property value is evaluated just before building the view corresponding to the subject of the selector:

1. The *value* arguments are converted to strings and concatenated together.
2. The result of the evaluation is a string which is parsed as a property value.

Example 1 (XHTML), simple table formatting could be implemented using this feature:

```
td, th {
  display: table-cell;
  text-align: concatenate(attr(align));
  vertical-align: concatenate(attr(valign));
  row-span: concatenate(attr(rowspan));
  column-span: concatenate(attr(colspan));
  border: 1 inset gray;
  padding: 2;
}
```

Example 2 (custom DTD) image name is the concatenation of a basename obtained from attribute *name* and an extension obtained from attribute *format* (see above to have a description of pseudo-function `image()` [47]):

```
image {
  content: concatenate("image('", attr(name), ".", attr(format), "',-400,-200)");
}
```

### 7.2. Using custom code to extend the CSS style sheet

In the rare cases where Cascading Style Sheets (CSS) are not powerful enough to style certain elements of a class of documents, it is possible to use custom code written in the Java™ language to do so.

`@extension "extension_class_name parameter ... parameter";` must be used to declare the Java™ class implementing the CSS extension.

Example (XHTML):

```
@extension "com.xmlmind.xmleditapp.xhtml.TableSupport black 'rgb(238, 238, 224)'"
```



In the above example, `com.xmlmind.xmlmleditapp.xhtml.TableSupport` is a class which is used to style XHTML (that is, HTML 4) tables. The two parameters which follow the class name specify colors used to draw table and cell borders. Parameters are optional and can be quoted if they contain white spaces.

The same CSS style sheet can contain several `@extension` constructs. For example, an extension class may be used to style HTML tables and another extension class may be used to localize generated content. If two `@extensions` reference the *same class name*, the last declared one will be used by the CSS engine. For example, redeclaring an extension class imported from another CSS style sheet may be useful to change its parameters.

How to write such extension class is explained in detail in the Chapter 8, *Writing style sheet extensions in XMLmind XML Editor - Developer's Guide*.

- The code of the extension class (contained in a `.jar` file) must have been loaded at XXE. This is done simply by copying the `.jar` file anywhere in one of the two `addons/` directories scanned by XXE during its startup. More information in Section 1, “Dynamic discovery of add-ons” in *XMLmind XML Editor - Configuration and Deployment*.
- Each time the style sheet containing the `@extension` rule is loaded, a new instance of the extension class is created.
- The extension class does not need to implement any specific interface but it must have a constructor with the following signature: `Constructor(java.lang.String[] parameters, com.xmlmind.xmlmledit.styledview.StyledViewFactory factory)`.
- Invoking the constructor of the extension class may have side effects such as registering intrinsic style specifications (`com.xmlmind.xmlmledit.stylesheet.StyleSpecs`, see Chapter 8, *Writing style sheet extensions in XMLmind XML Editor - Developer's Guide*) with the `com.xmlmind.xmlmledit.styledview.StyledViewFactory` passed as the second argument of the constructor.

The extension class may have methods which have been written to dynamically evaluate property values. These methods are invoked using the following syntax: `invoke(method_name, parameter, ..., parameter)`. Parameters are optional.

Example (Email schema used as an example in the Chapter 8, *Writing style sheet extensions in XMLmind XML Editor - Developer's Guide*):

```
from:before {
    content: invoke("localize", "From:");
}
```

In the above example, method `localize` of class `StyleExtension` is used to translate string "From:" to the language specified by the `xml:lang` attribute (if found on the email root element). For example, if `xml:lang` is set to `fr` (French), the generated content will contain "De:" instead of "From:".

Methods used to dynamically evaluate property values must have the following signature (see Chapter 8, *Writing style sheet extensions in XMLmind XML Editor - Developer's Guide*): `com.xmlmind.xmlmledit.stylesheet.StyleValue Method(com.xmlmind.xmlmledit.stylesheet.StyleValue[] parameters, com.xmlmind.xmlmledit.doc.Node contextNode, com.xmlmind.xmlmledit.styledview.StyledViewFactory factory)`.

If several extensions classes have dynamic property value methods with identical names (even if this unlikely to happen), the method actually used by the CSS engine will be the method of the class first declared using `@extension`.

## 8. New values for the display property

- **display: tree** may be used to mix styled element views and non-styled (tree-like) element views. This is particularly useful for meta-information (such as DocBook's `bookinfo`, `sectioninfo`, `indexterm`, etc) for which a sensible style is hard to come up with.

Example (DocBook):

The diagram illustrates the application of CSS table styling properties. It shows a tree structure of properties: **colspec** (with **align**: center, **colname**: 2, **colnum**: 2), **colspec** (with **colname**: 3), and **spanspec** (with **colsep**: 1, **nameend**: 3, **namest**: 2, **rowsep**: 1, **spanname**: 2x2). Below this, a table is shown with 4 columns and 4 rows. The cells contain coordinates (row, column) and a 2x2 spanned area in the center.

1, 1	1, 2	1, 3	1, 4
2, 1	2x2		2, 4
3, 1			3, 4
4, 1	4, 2	4, 3	4, 4

- **display: inline-block** may be used to specify a rectangular block that participates in an inline formatting context. (This is similar to **inline-table**.)

## 9. Rendering repeating elements as a table

In order to style repeating elements as a table, you need to have one element acting as the table (element `select` in the example below), one or more child elements acting as table rows (element `optgroup` in the example below) and one or more grandchild elements acting as table cells (element `option` in the example below), otherwise *it will not work*. Note that this limitation is in deviation with the CSS standard which requires the user agent to automatically generate anonymous tables, rows and cells when needed to.

- Two properties **column-span** and **row-span** have been added to specify the column and row span of elements with a **table-cell display**. The value for these properties is a strictly positive integer number. The initial value is 1. These properties are not inherited.
- The low-level property **start-column** is generally used by style sheet extensions to specify the start column of a cell in the case of complex tables. For example, this property is used by the Java™ code that styles DocBook/CALS tables. *Note that first column is column #0, not column #1.* The initial value is -1, which means the normal column for the cell. This property is not inherited.
- In addition to what is specified by CSS2, the **:before** and **:after** pseudo-elements allow values of the **display** property as follows:
  - If the subject of the selector is a **table** element, allowed values are **block**, **marker**, **table-row-group** and **table-row**. If the value of **display** has any other value, the pseudo-element will behave as if the value was **block**.
  - If the subject of the selector is a **table-row-group** element, allowed value is **table-row**. If the value of **display** has any other value, the pseudo-element will behave as if the value was **table-row**.
  - If the subject of the selector is a **table-row** element, allowed value is **table-cell**. If the value of **display** has any other value, the pseudo-element will behave as if the value was **table-cell**.

These extensions are supported to add generated column and row headers to arbitrary XML data displayed as a table.

For example, with these styles, the `select`, `optgroup` and `option` XHTML elements are displayed as a table with automatically generated column and row headers:

```
select {
  display: table;
  border: 1 solid black;
```

```
padding: 2;
border-spacing: 2;
background-color: silver;
}

select:before {
  display: table-row-group;
  content: row(cell("Category", width, 20ex), cell("Choice #1"),
               cell("Choice #2"), cell("Choice #3"),
               font-weight, bold, color, olive,
               padding-top, 2, padding-right, 2,
               padding-bottom, 2, padding-left, 2,
               border-width, 1, border-style, solid);
}

optgroup {
  display: table-row;
}

optgroup:before {
  display: table-cell;
  content: attr(label);
}

option {
  display: table-cell;
  border: 1 solid black;
  padding: 2;
  background-color: white;
}
```

XHTML source:

```
<select>
  <optgroup label="Language">
    <option>Java</option>
    <option>C++</option>
    <option>Perl</option>
  </optgroup>
  <optgroup label="Editor">
    <option>Emacs</option>
    <option>Vi</option>
    <option>UltraEdit</option>
  </optgroup>
  <optgroup label="OS">
    <option>Linux</option>
    <option>Windows</option>
    <option>Solaris</option>
  </optgroup>
</select>
```

Rendered as:

Category	Choice #1	Choice #2	Choice #3
Language	Java	C++	Perl
Editor	Emacs	Vi	UltraEdit
OS	Linux	Windows	Solaris

## 10. Making a table look like a spreadsheet

Use property **show-row-column-labels: yes/no** to add/remove A1-style labels to tables. Specify this property for elements with **display:table**, otherwise it is ignored.

Example: note that in DocBook, tgroup has **display:table**, not `table` or `informaltable`:

```
informaltable[role=spreadsheet] > tgroup {
  show-row-column-labels: yes;
}
```

	A	B
1	Investment	ROI
2		
3		

## 11. Collapsible blocks and tables

Elements with `display: block` or `display: table` can be made collapsible/expandable by specifying property `collapsible: yes`.

**Table 4.1. Properties used to parametrize the collapsibility of a block or table**

Property	Value	Initial value	Description
collapsible	yes   no	no	<p>yes block or table can be collapsed and expanded</p> <p>no block or table cannot be collapsed and expanded</p>
collapsed	yes   no	no	<p>yes block or table is initially collapsed</p> <p>no block or table is initially expanded</p>
not-collapsible-head	<i>non-negative integer</i>	0	Number of graphical items (gadgets) at the beginning of the block or table which must be kept visible even if the block or table is collapsed.
not-collapsible-foot	<i>non-negative integer</i>	0	Number of graphical items (gadgets) at the end of the block or table which must be kept visible even if the block or table is collapsed.
collapsed-content	<i>same as property content</i>	<i>no content</i>	<p>Content which must be displayed (in lieu of hidden graphical items) when the block or table is collapsed.</p> <p>Note that this content is transformed to an image before being used. Therefore this type of generated</p>

Property	Value	Initial value	Description
			<p>content cannot wrap at word boundaries.</p> <p><b>Caution</b></p> <p>Do not specify <code>label()</code> [51] or <code>indicator()</code> [45] as a value for <code>collapsed-content</code>. Instead specify <code>xpath()</code> [57].</p>
<code>collapsed-content-align</code>	<code>auto left center right</code>	<code>auto</code>	<p>Specifies how the <code>collapsed-content</code> image is to be horizontally aligned.</p> <p>Special value <code>auto</code> means that the <code>collapsed-content</code> image must be horizontally aligned just like the normal content it represents.</p>

The above properties cannot be inherited whether explicitly (**inherit** keyword) or implicitly (inherited property).

Examples:

```
section {
  collapsible: yes;
  not-collapsible-head: 1; /*keep title visible*/
}

table {
  collapsible: yes;
  not-collapsible-head: 1; /*keep title visible*/
  collapsed-content: url ../../icons2/table.gif);
}
```

Specifying `collapsible: yes` is not sufficient to be able to use collapsible blocks and tables. A special kind of toggle button called a *collapser* must be added to the generated content of the collapsible block or table or to the generated content of one of its descendants.

This toggle button is inserted in generated content using the `collapser()` pseudo-function [35].

Examples:

```
section > title:before {
  content: collapser() " " simple-counter(n-) " ";
}

table > title:before {
  content: collapser() " ";
}
```

The above examples show the most common case: A title or caption element is the mandatory first or last child of the collapsible block or table. This title or caption must always be kept visible (`not-collapsible-head: 1`). The collapser is inserted in the generated content of the title or caption.

The following example may be used to make a XHTML `div` collapsible. Note that a XHTML `div` has no mandatory first or last child. Therefore the collapser must be inserted in the generated content of the `div` itself.

```
div {
    display: block;
}

div[class=c3] {
    collapsible: yes;
}

div[class=c3]:before,
div[class=c3]:after {
    content: collapser();
    display: block;
    margin: 5 auto;
    text-align: center;
}

div[class=c3]:after {
    content: collapser(collapsed-icon, icon(collapsed-left),
                      expanded-icon, icon(expanded-up));
}
```

## Limitations

- A block, marked as being collapsible, can be collapsed only if it contains other blocks or tables. In the above example, an XHTML div of class c3 which just contains text *cannot* be collapsed.
- An element styled using "display:table;" is not collapsible per se. The table needs to contain a caption or title of any kind ("display:table-caption;") in order to be made collapsible.

In fact, only blocks containing other blocks or tables are potentially collapsible. Adding a caption to a table automatically creates an anonymous block containing both the caption and the table. It is this anonymous block which is collapsible.

## 12. Styling comments and processing instructions

The construct used for styling comments and processing instructions is similar to the standard construct used for styling the first line or the first letter of an element. Examples:

```
*:comment {
    background-color: yellow;
    display: block;
}

*:processing-instruction {
    background-color: green;
    display: block;
}

section > *:processing-instruction {
    content: icon(left-half-disc) processing-instruction-target() icon(right-half-disc);
    display: block;
}

para:processing-instruction/php {
    color: red;
    display: inline;
}
```

### Rule #1

specifies that comments are formatted as blocks with a yellow background.

The values allowed for the **display** property of comment and processing instruction pseudo-elements are: **inline**, **block**, **inline-block**.

## Rule #2

specifies that processing instructions are formatted as blocks with a green background.

Note that the target of the processing instruction is treated like a pseudo-attribute (editable using Edit → Processing Instruction → Change Processing Instruction Target) and is not considered to be part of its textual content.

## Rule #3

specifies that processing instructions which are contained in direct children of section have replaced content.

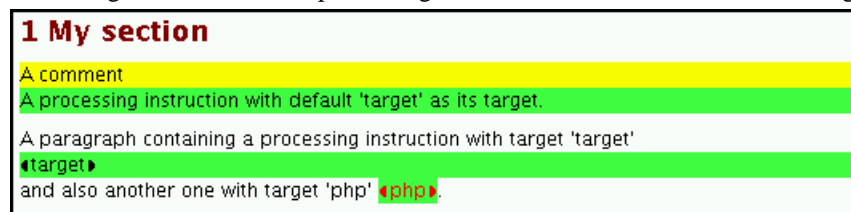
Comments and processing instructions may have replaced content but not generated content (**:before**, **:after**).

The replaced content of a processing instruction pseudo-element may contain **processing-instruction-target()** which is replaced by the target of the processing instruction subject of the rule.

## Rule #4

matches processing instructions with target "php" contained in `para` elements.

Rendering of comments and processing instructions in a DocBook article using the above style sheet:



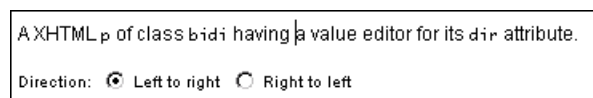
Note that it is also possible to use CSS3-like syntax `::comment` and `::processing-instruction` instead of CSS2-like syntax `:comment` and `:processing-instruction`.

## 13. Styling element attributes

An attribute can be rendered in the document view by inserting a value editor in the generated content.

XHTML example: a pair of radio buttons [53] are used to set the `dir` attribute of a `p` of class `bidirectional`.

```
p.bidi:after {
  display: block;
  content: "Direction: "
    radio-buttons(attribute, dir,
                  labels, "Left to right\A Right to left",
                  values, "ltr\A rtl");
  font-size: smaller;
}
```



This way of rendering attributes is fine but is too tedious to specify to be used on a large scale, for example to style XML data where most elements are empty but have several attributes.

In such case, it is recommended to use CSS rules where the selector contain the `:attribute()` non-standard pseudo-element.

The `:attribute()` pseudo-element has a attribute name parameter. This attribute name may be specified as a CSS identifier (specifies a name having no namespace, example: `dir`), a CSS string (also specifies a name having no namespace, example: `"dir"`) or a CSS qualified name (example: `xlink|href`). Note that name wildcards (examples: `xlink|*`, `*|href`) are not supported here.

Same example as above but using this type of rule:

```

p.bidi2:after {1
  display: block;
  content: attributes();
}

p.bidi2::attribute(dir) {2
  attribute-content-left: "Direction:";
  attribute-content-middle: radio-buttons(attribute, dir,
                                         labels,
                                         "Left to right\A Right to left",
                                         values, "ltr\A rtl");

  show-attribute: always;
  font-size: smaller;
}

```

- <sup>1</sup> First rule inserts an `attributes()` container [33] after each `p` of class `bidi2`.

A `attributes()` container is similar to a table with a row for each attribute. This table has 3 columns: left, middle, right. No border is drawn around its cells.

The content of an `attributes()` container is specified using CSS rules where the selector contain the `:attribute()` non-standard pseudo-element.

- <sup>2</sup> Second rule specifies that attribute `dir` must always be displayed for each `p` of class `bidi2`, whether this attribute is set or not.

`attribute-content-left` specifies the content of left column in the `attributes()` container. `attribute-content-middle` specifies the content of middle column in the `attributes()` container. `attribute-content-right` specifies the content of right column in the `attributes()` container.

**Table 4.2. Properties used to specify generated content for attributes**

Property	Value	Initial value	Description
<b>attribute-content-left</b>	Any value allowed for the <b>content</b> : property plus <code>attribute-*</code> () pseudo functions (see below [22]).	"" (no content)	Generated content for the attribute which is the target of the <code>:attribute()</code> rule that goes to the left column of the <code>attributes()</code> container.
<b>attribute-content-middle</b>	Any value allowed for the <b>content</b> : property plus <code>attribute-*</code> () pseudo functions (see below [22]).	"" (no content)	Generated content for the attribute which is the target of the <code>:attribute()</code> rule that goes to the middle column of the <code>attributes()</code> container.
<b>attribute-content-right</b>	Any value allowed for the <b>content</b> : property plus <code>attribute-*</code> () pseudo functions (see below [22]).	"" (no content)	Generated content for the attribute which is the target of the <code>:attribute()</code> rule that goes to the right column of the <code>attributes()</code> container.
<b>show-attribute</b>	<b>never</b>   <b>always</b>   <b>when-added</b>	<b>when-added</b>	never Never display this attribute in the <code>attributes()</code> container.  always Always display this attribute in the <code>attributes()</code> container even



Property	Value	Initial value	Description
			<p>if the attribute has not yet been added to the element.</p> <p>when-added Display this attribute in the <code>attributes()</code> container if the attribute has been added to the element.</p>

Same example as above with all attributes a `p` of class `bidir`, displayed when they are added to this element, except for the `dir` attribute which is always displayed:

```
p.bidi2:after {
  display: block;
  content: attributes();
}

p.bidi2::attribute() {1
  attribute-content-left: attribute-label() ":";2
  attribute-content-middle: 3value-editor(attribute, attribute());4
  attribute-content-right: remove-attribute-button(attribute, attribute());5
  show-attribute: when-added;
  font-size: smaller;
}

p.bidi2::attribute(dir) {6
  attribute-content-left: "Direction:";
  attribute-content-middle: radio-buttons(attribute, dir,
                                         labels,
                                         "Left to right\A Right to left",
                                         values, "ltr\A rtl");

  show-attribute: always;
}
```

A XHTML `p` of class `bidir` having a value editor for its `dir` attribute.

Class:  -

Direction: ☐ Left to right ☐ Right to left -

## Note

Notice that in the above figure, the values of the `dir` attribute are displayed in green. This is because, unlike in first example, this `p` of class `bidir` has no `dir` attribute yet.

By default (this can be specified [32]):

- A green foreground color means that attribute is not set.
- A red foreground color means that attribute value is invalid or that the value editor is not well suited to display this kind of values.

<sup>1</sup> This rule specifies the generated content for all attributes of a `p` of class `bidir`.

<sup>2</sup> `attribute-label()` is only supported in the `attribute-content-left`, `attribute-content-middle`, `attribute-content-right` properties.

Similar generated content is:

Pseudo-function	Description	Example
attribute-name()	The fully qualified name of the attribute.	ns:myAttribute-1
attribute-local-name()	Local part of attribute name.	myAttribute-1
attribute-namespace-uri()	Namespace URI of attribute name.	http://acme.com/ns/foo/bar
attribute-label()	Local part of attribute name, made more readable.	My attribute 1

- 3 value-editor() [57] will automatically find a suitable value editor based on the data type of attribute which is the target of the rule.
- 4 value-editor() like all other value editors (such as radio-buttons()) can also be used to edit the value of an element. "attribute, attribute()" specifies that the value editor to be inserted in generated content will be used to edit the attribute which is the target of the rule.
- 5 See remove-attribute-button() [54].
- 6 This rule specializes the previous rule for the dir attribute. The attribute-content-right property not specified in this rule is inherited from the more general :attribute() rule.

## 14. :property() extension pseudo class

Application properties are similar to element attributes except that:

- A property is not part of the document content.
- A property cannot be not saved to disk.
- The value of a property is not limited to a string but can be any Java™ object.
- The name of a property is an XML qualified name, just like for attributes.
- A property is not directly editable by the user. A property is added to an XML node by the application (that is, XEX).

Pseudo-function **property**(*property\_name*) can be used to insert the value of the property in generated content.

Pattern	Meaning
E:property(ns foo)	an E element, having a property named ns foo
E:property(ns foo, "bar") or E:property(ns foo, equals, "bar")	an E element, having a property named ns foo with a value whose string representation equals "bar"
E:property(ns foo, starts-with, "f")	an E element, having a property named ns foo with a value whose string representation starts with string "f".
E:property(ns foo, ends-with, "oo")	an E element, having a property named ns foo with a value whose string representation ends with string "oo".
E:property(ns foo, contains, "o")	an E element, having a property named ns foo with a value whose string representation contains substring "o".

An example of application property is {http://www.xmlmind.com/xmlmind/namespace/property}sourceURL, the location of the file from which an XML node has been loaded. Excerpt of XEX\_install\_dir/addon/config/common/css/visible\_inclusions.imp:

```
@namespace prop "http://www.xmlmind.com/xmlmind/namespace/property";

*:property(prop|sourceURL):before {
    content: icon(left-half-disc)
             "sourceURL=" property(prop|sourceURL)
             icon(right-half-disc);
    display: inherit;
```

```
color: red;
font-size: small;
text-align: center;
}
```

The above rule inserts above any element having a `{http://www.xmlmind.com/xmleditor/namespace/property}sourceURL` property, a block or an inline displaying the value of this property.

Another example is property `{http://www.xmlmind.com/xmleditor/namespace/property}readOnly`. This property is set on all XML nodes which cannot be modified by the user (e.g. included nodes). Excerpt of the *built-in* CSS file (automatically included before all the other CSS files):

```
@namespace prop "http://www.xmlmind.com/xmleditor/namespace/property";

*:property(prop|readOnly, "true") {
    background-color: #E0F0F0;
}

*:property(prop|readOnly, "false") {
    background-color: normal;
}
```

## 15. url() is XML catalog aware

The URI specified using the standard `url()` pseudo-function may be resolved using XML catalogs.

For example, this feature can be used to customize the DocBook CSS style sheet bundled with XXE:

```
@import url(xxe-config:docbook/css/docbook.css);❶

.
.
    my customization here
.
```

❶ Note that `@import "xxe-config:docbook/css/docbook.css";` works fine too. That is, in the case of `@import`, the `url()` pseudo-function is not strictly necessary for the XML catalogs to be used to resolve the URI.

This works because the XML catalog bundled with XXE, `XXE_install_dir/addon/config/catalog.xml`, contains the following rule:

```
<rewriteURI uriStartString="xxe-config:" rewritePrefix="." />
```

## 16. Modularizing a complex CSS style sheet using @property-group and @property-value

These extensions are useful when writing complex, modular, CSS style sheets. `@property-value` is especially useful when generating complex content such as embedded form controls.

### 16.1. @property-group

`@property-group` allows to define a named, possibly parametrized, group of properties. The syntax for defining such group is:

```
@property-group groupName( param1, ..., paramN ) {
    property;
    .
    .
    .
    property;
}
```

Including a `@property-group` in a rule is possible by using the following syntax:

```
selector {
    property;
    .
    .
    .
    property-group: groupName( argument1, ..., argumentN );
    .
    .
    .
    property;
}
```

Simple example:

```
@property-group title-style() {
    color: #004080;
    font-weight: bold;
}

@property-group standard-vmargins() {
    margin: 1.33ex 0;
}

title,
subtitle,
titleabbrev {
    display: block;
    property-group: title-style();
    property-group: standard-vmargins();
}
```

The above example is equivalent to:

```
title,
subtitle,
titleabbrev {
    display: block;
    color: #004080;
    font-weight: bold;
    margin: 1.33ex 0;
}
```

A @property-group can include other @property-groups. Example:

```
@property-group verbatim-style() {
    font-family: monospace;
    font-size: 0.83em;
}

@property-group verbatim-block-style() {
    display: block;
    white-space: pre;
    property-group: verbatim-style();
    property-group: standard-vmargins();
    border: thin solid gray;
    padding: 2px;
}

programlisting {
    property-group: verbatim-block-style();
}
```

The above example is equivalent to:

```
programlisting {
    display: block;
    white-space: pre;
    font-family: monospace;
    font-size: 0.83em;
}
```

```
margin: 1.33ex 0;
border: thin solid gray;
padding: 2px;
}
```

@property-groups can have formal parameters. When a @property-group is included in a rule, these formal parameters are replaced by actual arguments. Example:

```
@property-group verbatim-block-style(border-color) {
  display: block;
  white-space: pre;
  property-group: verbatim-style();
  property-group: standard-vmargins();
  border: thin solid border-color;
  padding: 2px;
}

programlisting {
  property-group: verbatim-block-style(#E0E0E0);
}
```

The above example is equivalent to:

```
programlisting {
  display: block;
  white-space: pre;
  font-family: monospace;
  font-size: 0.83em;
  margin: 1.33ex 0;
  border: thin solid #E0E0E0;
  padding: 2px;
}
```

A @property-group can even include a reference to itself. This simply means that the new definition extends (or partly overrides) the old one. Example:

```
@property-group verbatim-block-style(border-color, background-color) {
  property-group: verbatim-block-style(border-color);
  background-color: background-color;
}

programlisting {
  property-group: verbatim-block-style(rgb(127,127,127), #EEEEEE);
}
```

The above example is equivalent to:

```
programlisting {
  display: block;
  white-space: pre;
  font-family: monospace;
  font-size: 0.83em;
  margin: 1.33ex 0;
  border: thin solid rgb(127,127,127);
  padding: 2px;
  background-color: #EEEEEE;
}
```

## 16.2. @property-value

@property-value allows to defined a named, possibly parametrized, property value. The syntax for defining such named property value is:

```
@property-value name( param1, ..., paramN ) value ... value;
```

Including a @property-value in a property is possible by using the usual pseudo-function syntax:

```
propertyName: value ... name( argument1, ..., argumentN ) ... value;
```

Simple example:

```
@property-value generated-icon-color() gray;

indexterm:after {
    content: icon(right-half-disc);
    color: generated-icon-color();
}

anchor {
    content: icon(right-target);
    color: generated-icon-color();
}
```

The above example is equivalent to:

```
indexterm:after {
    content: icon(right-half-disc);
    color: gray;
}

anchor {
    content: icon(right-target);
    color: gray;
}
```

A @property-value can have formal parameters. When a @property-value is included in a property, these formal parameters are replaced by actual arguments. Example:

```
@property-value attributes-editor(margin, bg)
    attributes(margin-top, margin,
               margin-bottom, margin,
               margin-left, margin,
               margin-right, margin,
               background-color, bg);

@namespace foo "http://foo.com/ns";

foo|target {
    content: attributes-editor(2, #C0E0E0);
}
```

The above example is equivalent to:

```
foo|target {
    content: attributes(margin-top, 2,
                       margin-bottom, 2,
                       margin-left, 2,
                       margin-right, 2,
                       background-color, #C0E0E0);
}
```

Using the argument-list() pseudo-function, it is possible to replace a single formal parameter by a sequence of several actual arguments. Example:

```
foo|target {
    content: attributes-editor(2, argument-list(#C0E0E0, color, navy));
}
```

The above example is equivalent to:

```
foo|target {
    content: attributes(margin-top, 2,
                       margin-bottom, 2,
                       margin-left, 2,
                       margin-right, 2,
                       background-color, #C0E0E0,
                       color, navy);
}
```

The `argument-list()` pseudo-function may have no arguments at all, which is sometimes useful to suppress a formal parameter. Example:

```
@property-value attributes-editor(margin, args)
  attributes(margin-top, margin,
             margin-bottom, margin,
             margin-left, margin,
             margin-right, margin,
             args);

@namespace bar "http://bar.com/ns";

bar|target {
  content: attributes-editor(2, argument-list());
}
```

The above example is equivalent to:

```
bar|target {
  content: attributes(margin-top, 2,
                     margin-bottom, 2,
                     margin-left, 2,
                     margin-right, 2);
}
```

A `@property-value` can include other `@property-values`. Example:

```
@property-value header(title, bg)
  division(content(paragraph(content(collapser(collapsed-icon,
                                              icon(pop-right),
                                              expanded-icon,
                                              icon(pop-down)), " ",
                                              title,
                                              replace-button(), " ",
                                              insert-before-button(), " ",
                                              insert-button(), " ",
                                              insert-after-button(), " ",
                                              convert-button(), " ",
                                              delete-button(), " ",
                                              add-attribute-button(
                                                check-has-attributes, yes,
                                                color, navy)),
                                              background-color, bg,
                                              padding-left, 4),
            attributes-editor(2, bg)));

@namespace xs "http://www.w3.org/2001/XMLSchema";

xs|schema > xs|complexType:before,
xs|schema > xs|simpleType:before {
  content: header(argument-list(element-name(), " "),
                  #C0E0E0);
}
```

The above example is equivalent to:

```
xs|schema > xs|complexType:before,
xs|schema > xs|simpleType:before {
  content: division(content(paragraph(content(collapser(collapsed-icon,
                                              icon(pop-right),
                                              expanded-icon,
                                              icon(pop-down)), " ",
                                              element-name(), " ",
                                              replace-button(), " ",
                                              insert-before-button(), " ",
                                              insert-button(), " ",
                                              insert-after-button(), " ",
                                              convert-button(), " ",
                                              delete-button(), " ",
                                              add-attribute-button(
                                                check-has-attributes, yes,
                                                color, navy)),
                                              background-color, bg,
                                              padding-left, 4),
            attributes-editor(2, bg)));
}
```

```
        add-attribute-button(  
            check-has-attributes, yes,  
            color, navy)),  
        background-color, #C0E0E0,  
        padding-left, 4),  
        attributes-editor(2, #C0E0E0));  
}
```

A `@property-value` can even include a reference to itself. This simply means that the new definition specializes the old one. Example:

```
@property-value header(bg)  
    header(argument-list(element-name(), " ",  
        label(attribute, name, font-weight, bold), " "),  
        bg);  
  
xs|schema > xs|element:before {  
    content: header(#E0C0C0);  
}
```

The above example is equivalent to:

```
xs|schema > xs|element:before {  
    content: division(content(paragraph(content(collapser(collapsed-icon,  
        icon(pop-right),  
        expanded-icon,  
        icon(pop-down))), " ",  
        element-name(), " ",  
        label(attribute, name,  
            font-weight, bold), " "  
        replace-button(), " ",  
        insert-before-button(), " ",  
        insert-button(), " ",  
        insert-after-button(), " ",  
        convert-button(), " ",  
        delete-button(), " ",  
        add-attribute-button(  
            check-has-attributes, yes,  
            color, navy)),  
        background-color, #E0C0C0,  
        padding-left, 4),  
        attributes-editor(2, #E0C0C0));  
}
```

## 17. marker-offset: fill

For a content generated at the beginning of an element, with `display: marker`, this property allows to align the generated content to the left.

For a content generated at the end of an element, with `display: marker`, this property allows to align the generated content to the right.

Example (excerpts from DocBook's `structure.css`):

```
set:before,  
book:before,  
part:before,  
reference:before,  
preface:before,  
chapter:before,  
article:before,  
appendix:before,  
section:before,  
sect1:before,  
sect2:before,  
sect3:before,  
sect4:before,
```



```
sect5:before {
  display: marker;
  marker-offset: fill;
  content: element-name();
  font-size: small;
  color: gray;
}
```

## 18. If needed, selectors can use default attribute values

By default, as mandated in CSS2, attribute selectors only consider attributes explicitly specified for an element. However, it is possible to force attribute selectors to also consider default attribute values defined in the DTD, W3C XML Schema or RELAX NG schema. To do this, simply add "@use-default-attribute-values;" at the top of the CSS file.

DITA example:

```
@use-default-attribute-values;

*[class~="topic/body"] {
  display: block;
  margin-left: 12pt;
}
```

## 19. Simple, fast, purely declarative counters

Standard CSS counters, that is `counter-reset`, `counter-increment`, `counter()` and `counters()`, are fully supported by XXE. However, for most uses, we also have a simpler, faster because purely declarative, alternative to standard CSS counters.

Proprietary `simple-counter()` and `simple-counters()` may be used everywhere you use `counter()` and `counters()` and this, with a similar syntax: `simple-counters(n, ".")`, `simple-counter(n, upper-roman)`, etc. But, being purely declarative, you don't need to specify `simple-counter-reset` or `simple-counter-increment` in order to declare and update them.

Just like `counter` and `counters`, `simple-counter` and `simple-counters` are supported inside the `content` property. However their semantics are very different: *the name of the counter specifies the non-formatted value of the counter.*

Example 1 (XHTML):

```
ol > li:before {
  display: marker;
  content: simple-counter(n, decimal);
  font-weight: bold;
}
```

In the previous example, the counter name is `n` (single letter 'n', any letter is OK) which specifies that the counter value is the rank of `li` within its parent element (an `ol`).

Example 2 (DocBook):

```
sect3 > title:before {
  content: simple-counter(nnn-) " ";
}
```

In the previous example, the counter name is `nnn-` (3 letters followed by a dash) which specifies that the counter segmented value must be built as follows:

1. Skip (dash means skip) the rank of `title` within its parent element (a `sect3`).
2. Prepend (any letter means use) the rank of `title` parent (a `sect3`) within its parent (a `sect2`).

3. Prepend the rank of `title` grand-parent (a `sect2`) within its parent (a `sect1`).
4. Prepend the rank of `title` grand-grand-parent (a `sect1`) within its parent (an `article` or a `chapter`).

---

## Part II. Reference

---

---

## Chapter 5. Content objects

- `XXE_install_dir/demo/form-sampler.xml` is used to demo how standard controls such as buttons, check boxes, combo boxes, text fields, etc, can be embedded in the styled view. The CSS style sheets used by this demo are found in sub-directory `XXE_install_dir/demo/form-sampler/`.
- Most pseudo-functions create objects which can be styled at the object level. Styles are specified using *key*, *value* pairs where *key* is the name of a style property (example: **font-size**) and *value* is the value of a style property (example: **smaller**).

Example:

```
text-field(columns, 10,  
           background-color, white,  
           color, black)
```

- *Shorthand properties* cannot be used to specify style parameters as described above.

Example: **padding-top**, **padding-left**, **padding-bottom**, **padding-right** must be used rather than the single shorthand property **padding**.

- The above example is *conceptually* equivalent to (illegal CSS):

```
{ text-field(columns, 10);  
  background-color: white;  
  color: black; }
```

It is important to keep this in mind because it explains why you can specify:

```
text-field(columns, attr(cols),  
           background-color, white,  
           color, black)
```

but not:

```
text-field(columns, 10,  
           background-color, white,  
           color, attr(background))
```

The `attr()` construct can only be used in the value of property `content`: therefore it is not possible to specify `"color: attr(background);"`.

- All pseudo-functions generating controls (text-field [56], list [51], etc) also support the following color specifications:

Key	Value	Default	Description
<b>missing-color</b>	Color	rgb(0,128,128)	Foreground color used by the control when attribute or element value is missing. Therefore, this color is used when drawing default value.
<b>missing-background-color</b>	Color	None (no special background color)	Background color used by the control when attribute or element value is missing. Therefore, this color is used when drawing default value.
<b>error-color</b>	Color	rgb(128,0,0)	Foreground color used by the control when attribute or element value is invalid or when control is not well suited to edit this kind of value.
<b>error-background-color</b>	Color	None (no special background color)	Background color used by the control when attribute or element value is invalid or when

Key	Value	Default	Description
			control is not well suited to edit this kind of value.

Example:

```
text-field(columns, 10,
           missing-color, gray)
```

- All pseudo-functions generating content (except `icon()` [44] and `xpath()` [57]) accept `attr()` and `xpath()` values as well as literal values for their parameters.

Example:

```
text-field(columns, 10)
text-field(columns, attr(cols))
text-field(columns, xpath("5 + 5"))
```

- Most pseudo-functions are shorthand notations for `gadget(interface_name)`. See `gadget` [44].

For example, `collapser()` is a shorthand notation for `gadget("com.xmlmind.xmledit.form.Collapser")`, `command-button()` is a shorthand notation for `gadget("com.xmlmind.xmledit.form.CommandButton")`, etc.

## 1. add-attribute-button

Inserts a command-button [36] in generated content which can be used to add an attribute to the element for which the button has been generated.

Key	Value	Default	Description
<b>check-has-attributes</b>	Boolean: yes no, 1 0, true false, on off	no	<b>yes</b> means: do not generate this button when the target element has no attributes (attribute wildcards and <code>xsi:*</code> attributes are ignored).
<b>use-default-attribute-value</b>	Boolean: yes no, 1 0, true false, on off	no	<b>yes</b> means: newly added attribute is given its default value (as specified in the schema) if any, "???" otherwise.

Do not specify **command**, **parameter** or **menu** parameters for this type of command-button. A menu of `putAttribute` commands is built dynamically each time this button is clicked.

By default, this button has its icon set to `icon(plus)`.

Examples:

```
add-attribute-button()
add-attribute-button(text, "Add attr.",
                    check-has-attributes, yes,
                    use-default-attribute-value, yes)
```

## 2. attributes

**attributes**(key, value, ..., key, value)

Inserts in generated content a special purpose container. This special purpose container is populated with generated content for element attributes specified using **:attribute()** rules. See styling element attributes [19].

A `attributes()` container is similar to a table with a row for each attribute. This table has 3 columns: left, middle, right. No border is drawn around its cells.

Key	Value	Default	Description
<b>wrap-rows</b>	Boolean: yes no, 1 0, "true" "false", "on" "off"	yes	Specifies whether the rows of this tabular container are wrapped or not when they are too wide for the document view.

*Key, value, ..., key, value* may specify optional style parameters [32].

Examples:

```
attributes()
attributes(margin-top, 2,
           margin-bottom, 2,
           margin-left, 2,
           margin-right, 2)
attributes(wrap-rows, no)
```

### 3. check-box

**check-box**(*key, value, ..., key, value*)

Inserts a check box control in generated content. This control can be used to edit the value of the element which is the target of the CSS rule. If "**attribute**, *attribute\_name*" is specified, this control can be used to edit the value of an attribute of this target element.

Key	Value	Default	Description
<b>attribute</b>	Qualified name of attribute to be edited	No default	Without this parameter, the control is used to edit the value of the element for which the control has been generated.
<b>label</b>	String	None	Label used for the check box.
<b>unchecked-value</b>	String	None	In normal mode, unchecking the control assigns this value to the attribute or element value being edited.
<b>checked-value</b>	String	None	In normal mode, checking the control assigns this value to the attribute or element value being edited.
<b>remove-value</b>	Boolean: yes no, 1 0, "true" "false", "on" "off"	no	<p>Turns remove value mode on and off.</p> <p>In remove value mode, if unchecked-value is not specified, unchecking the control removes the attribute being edited.</p> <p>In remove value mode, if checked-value is not specified, checking the control removes the attribute being edited.</p> <p>If the value being edited is an element value rather than an attribute, this value is set to the empty string.</p>

*Key, value, ..., key, value* may also specify style parameters [32].

Examples:

```

check-box(attribute, value,
          label, "On",
          unchecked-value, "false",
          checked-value, "true")

check-box(label, "Yes",
          unchecked-value, "no",
          checked-value, "yes")

check-box(attribute, value,
          label, "Disabled",
          checked-value, "disabled",
          remove-value, yes)

check-box(label, "Hidden",
          checked-value, "hidden",
          remove-value, yes)

```

## 4. collapse

**collapse**(key, value, ..., key, value)

Inserts a toggle button in generated content which can be used to collapse a collapsible block or table. See collapsible blocks and tables [16].

Key	Value	Default	Description
<b>collapsed-icon</b>	url(), disc, circle, square, icon()	icon(collapsed-right)	Icon of the toggle button when the block or table is collapsed.
<b>expanded-icon</b>	url(), disc, circle, square, icon()	icon(expanded-down)	Icon of the toggle button when the block or table is expanded.
<b>tool-tip</b>	String	No default	Tool tip text of the toggle button.  A text starting with string "<html>" is understood to be styled using HTML ( <i>not XHTML</i> ). Example: "<html>Change the  <b>linkend</b>  attribute".

Key, value, ..., key, value may also specify style parameters [32].

Examples:

```

collapse()

collapse(tool-tip,
         "This is my tool tip!")

collapse(tool-tip,
         "<html>This is<br><b>my</b>tool tip!")

collapse(collapsed-icon, icon(pop-right),
         expanded-icon, icon(pop-down),
         color, navy)

```

## 5. combobox

**combobox**(key, value, ..., key, value)

Inserts a combobox control in generated content. This control can be used to edit the value of the element which is the target of the CSS rule. If "**attribute**, *attribute\_name*" is specified, this control can be used to edit the value of an attribute of this target element.

Key	Value	Default	Description
<b>attribute</b>	Qualified name of attribute to be edited	No default	Without this parameter, the control is used to edit the value of the element for which the control has been generated.
<b>labels</b>	List of strings separated by new lines ("A ")	None (use values as labels)	Labels used for the combobox items. The order of labels must match the order of values.
<b>values</b>	List of strings separated by new lines ("A ")	None ( <i>dynamically determined by examining the data type of value to be edited</i> )	Clicking on combobox item #N sets the element or attribute value being edited to value string #N.

*Key, value, ..., key, value* may also specify style parameters [32].

Examples:

```

combo-box(attribute, value)

combo-box(labels, "GreenA BlueA Red",
            values, "greenA blueA red")

```

## 6. command-button

**command-button**(*key, value, ..., key, value*)

Inserts a button in generated content which can be used to execute a command (see Chapter 6, *Commands written in the Java™ programming language in XMLmind XML Editor - Commands*) and/or to popup a menu of commands.

Key	Value	Default	Description
<b>icon</b>	url(), disc, circle, square, icon()	No default	Icon of the button. A button can have both a label and an icon.
<b>text</b>	String	No default	Label of the button. May contain newlines ("A " in CSS).  Element-*() pseudo functions are allowed here (see element-name [11]).
<b>tool-tip</b>	String	No default	Tool tip text of the button.  A text starting with string "<html>" is understood to be styled using HTML ( <i>not XHTML</i> ). Example: "<html>Change the  <b>linkend</b>  attribute".
<b>command</b>	String	No default	Name of command triggered by the button.
<b>parameter</b>	String	No default	Parameter of command triggered by the button.
<b>menu</b>	A menu of commands. See syntax below	No default	Menu of commands triggered by the button. A button can have both a command (Click-1) and a menu (Click-3).
<b>icon-gap</b>	Length (5px, 3em, etc)	4px	Distance between icon and label.
<b>icon-position</b>	right   top   bottom   left	left	Position of icon relative to the label.
<b>select</b>	none   element	element	By default, clicking a button selects the element having the generated content before at-



Key	Value	Default	Description
			tempting to execute the command. "select, none" disables this behavior.

*Key, value, ..., key, value* may also specify style parameters [32].

Menu syntax:

```
menu -> 'menu(' item+ ' )'

item->  label ',' command ',' parameter|'0'
      |  label ',' 'menu' ',' menu
      |  EMPTY_STRING ',' EMPTY_STRING ',' EMPTY_STRING
```

- 0 specifies a null parameter.
- " ", " ", " " is a separator.

Examples:

```
command-button(text, "Say hello",
               command, "alert",
               parameter, "Hello!",
               select, none,
               font-style, italic)

command-button(icon, icon(pop-right),
               menu, menu("Insert tr Before",
                           "insertNode", "sameElementBefore",
                           "Insert tr After",
                           "insertNode", "sameElementAfter",
                           " ", " ", " ",
                           "Delete tr", "delete", 0,
                           " ", " ", " ",
                           "Clipboard", menu, menu(
                               "Copy tr", "copy", 0,
                               "Cut tr", "cut", 0,
                               "Paste Before tr", "paste", "before",
                               "Paste After tr", "paste", "after"))))

command-button(text, "+",
               icon, disc,
               icon-position, right,
               icon-gap, 0,
               command, "insertNode",
               parameter, "sameElementAfter",
               menu, menu("Copy li", "copy", 0,
                           "Cut li", "cut", 0,
                           "Paste Before li", "paste", "before",
                           "Paste After li", "paste", "after"))
```

## 7. component

**component**(*className, param, ..., param*)

Inserts a standard Java™ AWT Component or Swing JComponent in generated content.

*className* is the name of a Java class which implements the interface `com.xmlmind.xmledit.styledview.ComponentFactory` (see Chapter 8, *Writing style sheet extensions in XMLmind XML Editor - Developer's Guide*).

Example (XHTML - excerpt of bundled `xhtml-form.css`):

```
input {
  content: component("com.xmlmind.xmleditapp.xhtml.Input");
}
```

## 8. convert-button

Inserts a command-button [36] in generated content which can be used to convert the element for which the button has been generated.

Do not specify **command**, **parameter** or **menu** parameters for this type of command-button. A menu of `convert` commands is built dynamically each time this button is clicked.

By default, this button has its icon set to `icon(convert)`.

Example:

```
convert-button()
```

## 9. date-field

**date-field**(*key*, *value*, ..., *key*, *value*)

Inserts in generated content a text field control, configured for parsing and formatting dates. This control can be used to edit the value of the element which is the target of the CSS rule. If "attribute, attribute\_name" is specified, this control can be used to edit the value of an attribute of this target element.

A `date-field` is used to convert a date specified using a normal, localized, format to/from a standard format. For example, the user sees and types something like "03/16/60" in the field (pattern is "MM/dd/YY") and the date actually stored in the XML document is "1960-03-16".

The date stored in the XML contains no time zone information, unless the date format specified by the `pattern` option contains a `z` or `Z` field. Example: the user sees and types something like "60/03/16 02:15 PM +0100" in the field (pattern is "YY/MM/dd hh:mm a Z") and the date actually stored in the XML document is "1960-03-16T13:15:00Z".

### Important

The `date-field` just converts a date format to another. The `date-field` is *not* used to validate what the user has typed. As always, the schema of the document is used to perform this validation. Therefore, beware that, when used with a DTD (which unlike W3C XML Schema or RELAX NG does not support data typing), a `date-field` allows the user to input incorrect date/time values.

See also `date-picker` [41].

Key	Value	Default	Description
<b>attribute</b>	Qualified name of attribute to be edited	No default	Without this parameter, the control is used to edit the value of the element for which the control has been generated.
<b>columns</b>	Positive integer	Depends on <code>pattern</code> .	Width of the text field in characters.
<b>pattern</b>	Pattern supported by <code>java.text.SimpleDateFormat</code>	A simple pattern which depends on <code>data-type</code> .	Specifies how date is to be parsed and formatted.
<b>language</b>	Lower-case, two-letter codes as defined by ISO-639. Example: "es".	Language of default locale.	Participates in specifying the locale to use.
<b>country</b>	Upper-case, two-letter codes as defined by ISO-3166. Example: "ES".	Country of default locale.	Participates in specifying the locale to use.

Key	Value	Default	Description
<b>variant</b>	Vendor or browser-specific code. Example: "Traditional_WIN".	Variant of default locale.	Participates in specifying the locale to use.
<b>data-type</b>	date   time   dateTime   gDay   gMonthDay   gMonth   gMonthYear   gYear	date	Base data type of attribute or element value being edited.  Note that default pattern for <b>gMonthDay</b> is MM/dd and default pattern for <b>gYearMonth</b> is yyyy/MM.

*Key, value, ..., key, value* may also specify style parameters [32].

Example:

```
date-field()

date-field(pattern, "yy/MM/dd hh:mm a Z",
           data-type, dateTime,
           language, en,
           country, "US")
```

## 10. date-time-picker

**date-time-picker**(*key, value, ..., key, value*)

Inserts in generated content a text field control and a button which displays a dialog box allowing to select a date/time. This control can be used to edit the value of the element which is the target of the CSS rule. If "attribute, attribute\_name" is specified, this control can be used to edit the value of an attribute of this target element.

Unlike what happens with a date-field [38], the *same date format* is used to display the value on screen and to store it in the XML document. This allows to use a date-time-picker for data types other than those deriving from xs:dateTime.

Example:

```
<xs:element name="dateTime31" type="DateTime31" />
...
<xs:complexType name="DateTime31">
  <xs:attribute name="value" type="DateTime41" default="1315 03/16/1960" />
</xs:complexType>

<xs:simpleType name="DateTime41">
  <xs:restriction base="xs:token">
    <xs:pattern value="\d\d\d\d \d\d/\d\d/\d\d\d\d" />
  </xs:restriction>
</xs:simpleType>
```

may be edited using:

```
dateTime31 {
  content: date-time-picker(attribute, value,
                           format, pattern,
                           pattern, "HHmm MM/dd/yyyy",
                           language, en,
                           country, "US");
}
```

Key	Value	Default	Description
<b>attribute</b>	Qualified name of attribute to be edited	No default	Without this parameter, the control is used to edit the value of the element for which the control has been generated.
<b>columns</b>	Positive integer	Depends on <code>format</code> and on <code>pattern</code> .	Width of the text field in characters.
<b>format</b>	<code>standard</code>   <code>standard-omit-time-zone</code>   <code>pattern</code>   <code>seconds-since-epoch</code>   <code>millis-since-epoch</code>	<code>standard-omit-time-zone</code>	<p>Specifies the format of the date/time value.</p> <p><b>standard</b> Standard format of a <code>xs:dateTime</code> (W3C XML Schema standard data type), with the time zone part.  Example: 2001-09-11T11:30:00Z.</p> <p><b>standard-omit-time-zone</b> Standard format of a <code>xs:dateTime</code> (W3C XML Schema standard data type), without the time zone part.  Example: 2007-10-18T09:16:26.</p> <p><b>pattern</b> The format of the date/time value is the one specified by the <code>pattern</code> option (see below).  Example: 1415 03/16/1960 (for <code>pattern="HHmm MM/dd/yyyy"</code>)</p> <p><b>seconds-since-epoch</b> Number of seconds since January 1, 1970 GMT. A real number (<code>double</code>).  Example: 1.192699294E9.</p> <p><b>millis-since-epoch</b> Number of milliseconds since January 1, 1970 GMT. An integer (<code>long</code>).  Example: 1192699313795.</p>
<b>pattern</b>	Pattern supported by <code>java.text.SimpleDateFormat</code>	A default, short, pattern depending on the locale being used.	<p>Specifies the format of the date.</p> <p>Ignored unless <code>format=pattern</code>.</p>
<b>language</b>	Lower-case, two-letter codes as defined by ISO-639. Example: "es".	Language of default locale.	Participates in specifying the locale to use.
<b>country</b>	Upper-case, two-letter codes as defined by ISO-3166. Example: "ES".	Country of default locale.	Participates in specifying the locale to use.
<b>variant</b>	Vendor or browser-specific code. Example: "Traditional_WIN".	Variant of default locale.	Participates in specifying the locale to use.

*Key, value, ..., key, value* may also specify style parameters [32].

Examples:

```
dateTime30 {
    content: date-time-picker(attribute, value,
                               format, standard,
                               columns, 25);
}

dateTime41 {
    content: date-time-picker(format, pattern,
                               pattern, "HHmm MM/dd/yyyy",
                               language, en,
                               country, "US");
}

dateTime43 {
    content: date-time-picker(format, millis-since-epoch);
}
```

## 11. date-picker

Similar to date-time-picker [39], except that the dialog box displayed by the button allows to select a date (and not a date/time).

Examples:

```
date30 {
    content: date-picker(attribute, value);
}

date41 {
    content: date-picker(format, pattern,
                          pattern, "MM/dd/yyyy",
                          language, en,
                          country, "US");
}
```

## 12. delete-button

A convenient way of specifying `command-button` [36](`icon, icon(delete), command, "delete", parameter, 0`).

## 13. drag-source

**drag-source**(*key, value, ..., key, value*)

Inserts a button in generated content which can be used to execute a command (see Chapter 6, *Commands written in the Java™ programming language in XMLmind XML Editor - Commands*). Identical to `command-button` [36] except that:

- A `drag-source` cannot be used to popup a menu.
- The user cannot click on a `drag-source`. He/she needs to drag the mouse over it to trigger the command. *This command must return a string.*

Example:

```
section[id] > title:after {
    display: inline;
    content: drag-source(icon, icon(right-link),
                        command, "dragHref");
}
```

where command `dragHref` is:

```
<command name="dragHref">
  <macro xmlns:hrefu="java:com.xmlmind.xmleditapp.dita.HrefUtil">
    <sequence>
      <command name="selectNode" parameter="parent section" />
      <get expression="hrefu:get-href($selectedElement)" />
    </sequence>
  </macro>
</command>
```

## 14. drop-site

**drop-site**(*key, value, ..., key, value*)

Inserts a button in generated content which can be used to execute a command (see Chapter 6, *Commands written in the Java™ programming language in XMLmind XML Editor - Commands*). Identical to command-button [36] except that:

- A drop-site cannot be used to popup a menu.
- The user cannot click on a drop-site. He/she needs to drop a string (typically a filename or an URL coming from a file manager or a Web browser) on it to trigger the command.
- The parameter of the command must contain variable `%{value}` which is substituted with the dropped *string*.

If the object dropped from an external application is not a string (that is, some text), this object will be automatically converted to a string (when possible). For example, a file is converted to a string by using its absolute filename.

In addition to `%{value}`, the following convenience variables are also supported:

`%{url}`

If `%{value}` contains an URL or the absolute filename of a file or a directory, this variable contains the corresponding URL.

`%{file}`

If `%{value}` contains a "file:" URL or the absolute filename of a file or a directory, this variable contains the corresponding filename.

Example:

```
br|date:after {
  display: block;
  content: drop-site(text, "Drop a screen shot here",
    icon, url(drop.gif),
    icon-position, right,
    command, "paste",
    parameter, "after <?xml version='1.0'?><screenShot \
      xmlns='http://www.xmlmind.com/xmlmind/schema/bugreport' \
      image='%{value}' />");
}
```

## 15. file-name-field

**file-name-field**(*key, value, ..., key, value*)

Inserts in generated content both a text field control and a button which can be used to browse files. These controls can be used to edit the value of the element which is the target of the CSS rule. If "**attribute**, *attribute\_name*" is specified, these controls can be used to edit the value of an attribute of this target element.

Key	Value	Default	Description
<b>attribute</b>	Qualified name of attribute to be edited	No default	Without this parameter, the control is used to edit the value of the element for which the control has been generated.
<b>columns</b>	Positive integer	20	Width of the text field in characters.
<b>absolute</b>	Boolean: yes no, 1 0, "true" "false", "on" "off"	no	Configures the file chooser dialog box.  yes Dialog box returns an absolute path.  no Dialog box returns a path which is relative to the entity containing the target element (when possible).
<b>directory</b>	Boolean: yes no, 1 0, "true" "false", "on" "off"	no	Configures the file chooser dialog box.  yes Dialog box can only select directories.  no Dialog box can only select files.
<b>save</b>	Boolean: yes no, 1 0, "true" "false", "on" "off"	no	Configures the file chooser dialog box.  yes Dialog box can select existing files or directories, as well as files and directories to be created.  no Dialog box can only select existing files or directories.
<b>url</b>	Boolean: yes no, 1 0, "true" "false", "on" "off"	yes	Configures the file chooser dialog box.  yes Dialog box returns URLs  no Dialog box returns file names.

*Key, value, ..., key, value* may also specify style parameters [32].

Examples:

```
file-name-field(attribute, value,
               columns, 40,
               font-family, monospaced)

file-name-field(absolute, yes,
               directory, yes,
               save, yes,
               url, no,
               columns, 40,
               font-family, monospaced)
```

## 16. gadget

**gadget**(*className*, *param*, ..., *param*).

This pseudo-function is similar to the component [37] pseudo-function except that it creates lightweight *gadgets* instead of standard Java™ AWT Components or Swing JComponents.

*className* is the name of a Java class which implements the interface `com.xmlmind.xmledit.styledview.GadgetFactory` (see Chapter 8, *Writing style sheet extensions in XMLmind XML Editor - Developer's Guide*).

Example (APT - excerpt of `apt-collapsible.css`):

```
caption:before {
  content: gadget("com.xmlmind.xmledit.form.Collapser",
    collapsed-icon, icon(collapsed-right),
    expanded-icon, icon(expanded-up)) " ";
}
```

When `gadget()` is used to generate replaced content for a processing-instruction, the specified class must implement interface `com.xmlmind.xmledit.styledview.GadgetFactory2` (see Chapter 8, *Writing style sheet extensions in XMLmind XML Editor - Developer's Guide*). Example, the following rule is used to style spreadsheet formulas:

```
*::processing-instruction(xxe-formula) {
  content: gadget("com.xmlmind.xmleditapp.spreadsheet.Formula");
  display: inline;
}
```





















## 17. icon


















**icon**(*name*, *foreground*, *background*, *tool\_tip*)
























Inserts a built-in image in generated content.

*name*

Required.

The name of the icon must be one of the following identifiers:  check-off,  check-on,  circle,  collapsed-left,  collapsed-right,  convert,  delete,  diamond,  disc,  down,  drop2,  drop,  expand-down,  expanded-down,  expanded-up,  expand-up,  external-link,  external-link-small,  file,  filled-square,

 half-disc-separator,  hollow-diamond,  image,  image-small,  insert-after,  insert-before,  insert,  invisible,  launch,  left-half-disc,  left-link,  left,  left-target,  line-break,  link,  minus-box,  minus,

 no-image,  pin-down,  pin-left,  pin-right,  pin-up,  plus-box,  plus,  pop-down,  pop-left,  pop-ne,  pop-nw,  pop-right,  pop-se,  pop-sw,  pop-up,  radio-off,  radio-on,  replace,  return,  right-half-disc,  right-link,  right,  right-target,  square-3,  square-5,  square,  up.

*foreground*

Optional.

Specifies the foreground color of a *bitonal* icon (e.g. `diamond` and not `expand-down`). By default, the bitonal icons inherit their foreground color from the `color` property of the CSS style sheet.

### Tip

Specify something which is not a valid color (e.g. number 0) if you want to force the style sheet engine to use the current foreground color.

This tip also applies to the *background* parameter except that in this case, the current background color is used.



**background**  
Optional.

Specifies the background color of a bitonal icon. By default, the bitonal icons inherit their background color from the `background-color` property of the CSS style sheet.

**tool\_tip**  
Optional.

Specifies a tool tip text of the icon.

Examples:

```
icon(drop2)
icon(pop-right, red)
icon(pop-down, #333333, transparent)
icon(pin-right, 0, transparent, "Draft comments")
```

## 18. indicator

**indicator**(*key, value, ..., key, value*)

Inserts in generated content an image (taken from specified set of images) which is determined using the value of specified attribute or XPath expression.

Similar to `label` [51] except that `indicator` is rendered using a set of images rather than text.

Key	Value	Default	Description
<b>attribute</b>	Qualified name of an attribute of the element which is the target of the rule	No default One of <b>attribute</b> or <b>xpath</b> must be specified.	The value of this attribute is compared to the values of the <b>state</b> arguments. If one of the <b>state</b> argument is found equal to this value, the corresponding icon is displayed. Otherwise first icon is displayed.
<b>xpath</b>	Literal string specifying an XPath expression using the target of the rule as its context node	No default One of <b>attribute</b> or <b>xpath</b> must be specified.	The value of this XPath expression is compared to the values of the <b>state</b> arguments. If one of the <b>state</b> argument is found equal to this value, the corresponding icon is displayed. Otherwise first icon is displayed.
<b>state</b>	Identifier or string	No default	Specifies one of the states of the indicator.  Must be followed by corresponding <b>icon</b> argument.  An indicator always contains several <b>state/icon</b> pairs.
<b>icon</b>	<code>url()</code> , <code>disc</code> , <code>circle</code> , <code>square</code> , <code>icon()</code>	No default	Specifies one of the images used to render the indicator.  Corresponding <b>state</b> must precede this argument.  An indicator always contains several <b>state/icon</b> pairs.

*Key, value, ..., key, value* may also specify style parameters [32].

XHTML examples:

```
p.msg:before {
  content: indicator(attribute, title,
                    state, info, icon, url(info.gif),
                    state, warning, icon, url(warning.gif),
                    state, error, icon, url(error.gif));
  display: marker;
}

div.hotel span.with_stars:after {
  content: " "
    indicator(xpath,
              "substring-after(ancestor::div[@class='hotel']/@title,\
                              'stars')",
              state, "not Rated", icon, icon(diamond),
              color, gray,
              state, "0", icon, url(0star.gif),
              state, "1", icon, url(1star.gif),
              state, "2", icon, url(2star.gif),
              state, "3", icon, url(3star.gif),
              state, "4", icon, url(4star.gif),
              state, "5", icon, url(5star.gif));
  display: inline;
}
```

## 19. insert-after-button

Inserts a command-button [36] in generated content which can be used to insert an element or text node after the element for which the button has been generated.

Do not specify **command**, **parameter** or **menu** parameters for this type of command-button. A menu of "insert after" commands is built dynamically each time this button is clicked.

By default, this button has its icon set to `icon(insert-after)`.

Example:

```
insert-after-button()
```

## 20. insert-before-button

Inserts a command-button [36] in generated content which can be used to insert an element or text node before the element for which the button has been generated.

Do not specify **command**, **parameter** or **menu** parameters for this type of command-button. A menu of "insert before" commands is built dynamically each time this button is clicked.

By default, this button has its icon set to `icon(insert-before)`.

Example:

```
insert-before-button()
```

## 21. insert-button

Inserts a command-button [36] in generated content which can be used to insert an element or text node into the element for which the button has been generated.

Do not specify **command**, **parameter** or **menu** parameters for this type of command-button. A menu of insert commands is built dynamically each time this button is clicked.

By default, this button has its icon set to `icon(insert)`.

Example:

```
insert-button()
```

## 22. insert-same-after-button

A convenient way of specifying `command-button [36](icon, icon(insert-after), command, "insertNode", parameter, "sameElementAfter")`.

## 23. insert-same-before-button

A convenient way of specifying `command-button [36](icon, icon(insert-before), command, "insertNode", parameter, "sameElementBefore")`.

## 24. image

**image**(*source*, *width*, *height*, **smooth**|**default**, *fallback\_image*).

Inserts a user defined, possibly scaled, image in generated content.

*source*

Required.

URL or path of an image file. Only GIF, JPEG, PNG files will be displayed by XXE but this must not prevent you from using other formats if your backend processor supports them.

A relative URL or path is relative to the location of the document being edited and not to the current working directory.

*width, height*

Optional.

Dimension of the image in pixels. A length may optionally be followed by a standard CSS unit such as **px**, **in**, **cm**, **mm**, **pt**, **pc**, **em**, **ex**.

A negative length is interpreted as a maximum size. This is useful to display images as thumbnails.

**auto** specifies intrinsic image size.

**smooth**|**default**

Optional.

The name of the algorithm used to change the image size: **smooth** means high-quality/slow and **default** means low-quality/fast.

*fallback\_image*

Optional.

Specifies which fallback image to use when image specified by first argument cannot be loaded. All forms of image specification supported by XXE (except **image()**) can be used for this argument: **url()**, **icon()**, **circle**, etc.

Examples (XHTML):

```
img {
  content: image(attr(src));
}

img {
  content: image(attr(src), -600, -400);
}
```

```
img {
  content: image(attr(src), attr(width), attr(height), default, icon(no-image));
}
```

## 25. image-viewport

**image-viewport**(*key, value, ..., key, value*)

Inserts an image in generated content.

The image is displayed, possibly after being scaled, in a viewport (that is, a rectangle possibly larger than the displayed image).

This content object, functionally close to the XSL-FO `fo:external-graphic` element, is a sophisticated variant of `image()` [47].

Unless a **source** parameter is specified (see table below), the image-viewport is associated to an attribute or to an element (that is, the image-viewport is a “view” of the attribute or of the element). This attribute or this element may reference the URL of an external graphics file or may directly contain image data. In such case, the image-viewport object can also be used to edit this attribute or this element. To do this, the XXE user needs to double-click on the image-viewport and then specify a graphics file using a specialized dialog box. Alternatively the XXE user can also drag and drop a graphics file on the image-viewport.

Key	Value	Default	Description
<b>source</b>	<code>url()</code>	None. Image data comes from the element for which this image-viewport is a view.	Specifies the URL of graphics file to be displayed by the image-viewport.  Rarely used. Most image-viewport are associated to attributes or to elements.
<b>descendant</b>	String evaluated as an XPath expression returning a node-set	None. Image data comes from the element for which this image-viewport is a view.	Specifies a descendant element of the element for which this image-viewport is a view. Example: DocBook 5's <code>imagedata/svg:svg</code> or <code>imagedata/mml:math</code> .  Rarely used. Most image-viewport are associated to attributes or to elements.
<b>attribute</b>	Qualified name of attribute to be edited	None. Image data comes from the element for which this image-viewport is a view.	Specifies the name of the attribute containing the URL of graphics file to be displayed by the image-viewport ( <b>data-type=anyURI</b> ) or directly containing image data ( <b>data-type=hexBinary</b> or <b>base64Binary</b> ).
<b>data-type</b>	<code>anyURI   hexBinary   base64Binary   XML</code>	None. If the document is conforming to a W3C XML Schema or to a RELAX NG schema, this data-type can be found automatically. Otherwise (DTD, no grammar), specifying this parameter is mandatory.	Specifies how the image is “stored” in the attribute or in the element.  <b>data-type=XML</b> is only allowed for elements (typically an <code>svg:svg</code> element).
<b>gzip</b>	Boolean: <code>yes no, 1 0, "true" "false", "on" "off"</code>	no	Ignored unless <b>data-type=hexBinary</b> or <b>base64Binary</b> .

Key	Value	Default	Description
			If <b>true</b> , image data will be compressed with gzip before being encoded in <b>hexBinary</b> or in <b>base64Binary</b> .
<b>viewport-width</b>	Length (px, mm, em, etc) or percentage	None.	Specifies the width of the viewport.  A percentage (ex. 50%) is a percentage of the available space.
<b>viewport-height</b>	Length (px, mm, em, etc) or percentage	None.	Specifies the height of the viewport.  A percentage is a percentage of the available space. <i>This is currently not supported.</i>
<b>content-width</b>	Length (px, mm, em, etc) or percentage or scale-to-fit or a max. size	None.	Specifies the width of the image after rescaling it.  A percentage is a percentage of the intrinsic width.  <b>scale-to-fit</b> means change the width of the image to fit the viewport.  A <i>max. size</i> is specified like this: 200max. This means: at most 200 pixels. Therefore if the image is wider than 200 pixels, its width is scaled down to 200. Otherwise, the intrinsic width is used as is.
<b>content-height</b>	Length (px, mm, em, etc) or percentage or scale-to-fit or a max. size	None.	Specifies the height of the image after rescaling it.  A percentage is a percentage of the intrinsic height.  <b>scale-to-fit</b> means change the height of the image to fit the viewport.  A <i>max. size</i> is specified like this: 400max. This means: at most 400 pixels. Therefore if the image is taller than 400 pixels, its height is scaled down to 400. Otherwise, the intrinsic height is used as is.
<b>preserve-aspect-ratio</b>	Boolean: yes no, 1 0, "true" "false", "on" "off"	yes	Ignored unless <b>content-width</b> and <b>content-height</b> are both set to <b>scale-to-fit</b> or are both set to a <i>max. size</i> .  If <b>false</b> , the image is scaled non-uniformly (stretched) to fit the viewport.
<b>smooth</b>	Boolean: yes no, 1 0, "true" "false", "on" "off"	no	If <b>true</b> , quality is favored over speed when rescaling the image.
<b>horizontal-align</b>	left   center   right	center	Specifies how the image is to be horizontally aligned in the viewport.
<b>vertical-align</b>	top   middle   bottom	middle	Specifies how the image is to be vertically aligned in the viewport.
<b>fallback-image</b>	url(), disc, circle, square, icon()	Automatically generated. May contain an	Specifies which image to display when the normal image cannot be displayed (image

Key	Value	Default	Description
		error message displayed in red.	format not supported, file not found, corrupted image, etc)

*Key, value, ..., key, value* may also specify style parameters [32].

Simple example (XHTML):

```
img {
  display: inline;
  content: image-viewport(attribute, src,
                          data-type, anyURI,
                          content-width, attr(width),
                          content-height, attr(height));
}
```

Other example (DocBook 5, images displayed as 128x128 thumbnails):

```
@namespace svg "http://www.w3.org/2000/svg";

imagedata:contains-element(svg|svg) {
  content: image-viewport(descendant, "./svg:svg",
                          data-type, XML,
                          content-width, 128max,
                          content-height, 128max);
}
```

Complex example (ImageDemo, see *XXE\_install\_dir/doc/configure/samples/imagedemo*):

```
image_ab {
  /*
   * No need to specify data-type. The image-viewport will find it by itself.
   */

  content: image-viewport(attribute, data, gzip, true,
                          viewport-width, attr(width),
                          viewport-height, attr(height),
                          preserve-aspect-ratio, attr(preserve_aspect_ratio),

                          content-width,
                          xpath("if(@content_width='scale_to_fit',\
                                'scale-to-fit',\
                                @content_width)"),

                          content-height,
                          xpath("if(@content_height='scale_to_fit',\
                                'scale-to-fit',\
                                @content_height)"),

                          horizontal-align
                          xpath("if(@anchor='west' or @anchor='north_west' or @anchor='south_west',\
                                'left',\
                                @anchor='center' or @anchor='north' or @anchor='south',\
                                'center',\
                                @anchor='east' or @anchor='north_east' or @anchor='south_east',\
                                'right',\
                                'center')"),

                          vertical-align,
                          xpath("if(@anchor='north' or @anchor='north_east' or @anchor='north_west',\
                                'top',\
                                @anchor='center' or @anchor='east' or @anchor='west',\
                                'middle',\
                                @anchor='south' or @anchor='south_east' or @anchor='south_west',\
                                'bottom',\
                                'middle')"));
};
}
```

## 26. label

**label**(*key, value, ..., key, value*)

Inserts in generated content the value of specified attribute or XPath expression.

### Difference with standard construct **attr()** and with extension **xpath()** [57]:

**xpath()** and **attr()** are evaluated *once* and this happens when the view of the element is built. This means that in some cases, manually refreshing the view of the element after a change in the document will be needed (use Select → Redraw (**Ctrl+L**)).

Unlike **xpath()** and **attr()**, **label()** is automatically updated when the document is modified.

For efficiency reasons, the update of **label(xpath, XPath\_expression)** is delayed until the editing context changes.

Key	Value	Default	Description
<b>attribute</b>	Qualified name of an attribute of the element which is the target of the rule	No default One of <b>attribute</b> or <b>xpath</b> must be specified.	Display the value of this attribute as styled text.
<b>xpath</b>	Literal string specifying an XPath expression using the target of the rule as its context node	No default One of <b>attribute</b> or <b>xpath</b> must be specified.	Display the value of this XPath expression as styled text.

*Key, value, ..., key, value* may also specify style parameters [32].

XHTML examples:

```
p.msg:before {
  content: label(attribute, title,
                 text-decoration, underline);
  display: marker;
}

a.showtarget {
  content: icon(pop-right)
    label(xpath, "//*[name = substring-after(current()@href, '#')]",
           text-decoration, underline);
}

caption.formal:before {
  content: "Table "
    label(xpath, "1 + count(..preceding::table[caption])")
    ": ";
  display: inline;
}
```

See also **indicator** [45] which is similar to **label** except that **indicator** rendered using a set of images rather than text.

## 27. list

**list**(*key, value, ..., key, value*)

Inserts a list control in generated content. This control can be used to edit the value of the element which is the target of the CSS rule. If "**attribute**, *attribute\_name*" is specified, this control can be used to edit the value of an attribute of this target element.

Key	Value	Default	Description
<b>attribute</b>	Qualified name of attribute to be edited	No default	Without this parameter, the control is used to edit the value of the element for which the control has been generated.
<b>labels</b>	List of strings separated by new lines (" <i>\A</i> ")	None (use values as labels)	Labels used for the list items. The order of labels must match the order of values.
<b>values</b>	List of strings separated by new lines (" <i>\A</i> ")	None ( <i>dynamically determined by examining the data type of value to be edited</i> )	<p>In single selection mode, clicking on list item #N sets the element or attribute value being edited to value string #N.</p> <p>In multiple selection mode clicking on list item #N adds/removes value string #N to/from the selected set.</p> <p>The value strings in the selected set are then joined using the character specified by separator ( ' ' by default).</p> <p>The resulting string is assigned to the element or attribute value being edited.</p>
<b>rows</b>	Positive integer	max(10, number of values)	Maximum number of rows displayed by the list control.
<b>selection</b>	single   multiple	single	Specifies single or multiple selection mode.
<b>separator</b>	Single character string	None (values are separated by any type of white space characters)	Character used to join selected value strings in multiple selection mode. The resulting string is assigned to the element or attribute value being edited.

*Key, value, ..., key, value* may also specify style parameters [32].

Examples:

```
list(rows, 3)

list(attribute, value,
      labels, "Cyan\A Yellow\A Magenta\A Black")

list(rows, 3,
      selection, multiple)

list(attribute, value,
      labels, "Cyan\A Yellow\A Magenta\A Black",
      values, "cyan\A yellow\A magenta\A black",
      selection, multiple,
      separator, ",")
```

## 28. number-field

**number-field**(*key, value, ..., key, value*)

Inserts in generated content a text field control, configured for parsing and formatting numbers. This control can be used to edit the value of the element which is the target of the CSS rule. If "*attribute, attribute\_name*" is specified, this control can be used to edit the value of an attribute of this target element.



A `number-field` is used to convert a number specified using a normal, localized, format to/from a standard format. For example, the user sees and types something like "1000000000.0" in the field (pattern is "0.0#####") and the number actually stored in the XML document is "1.0E9".

## Important

The `number-field` just converts a number format to another. The `number-field` is *not* used to validate what the user has typed. As always, the schema of the document is used to perform this validation. Therefore, beware that, when used with a DTD (which unlike W3C XML Schema or RELAX NG does not support data typing), a `number-field` allows the user to input incorrect numbers.

Key	Value	Default	Description
<b>attribute</b>	Qualified name of attribute to be edited	No default	Without this parameter, the control is used to edit the value of the element for which the control has been generated.
<b>columns</b>	Positive integer	Depends on <code>pattern</code> .	Width of the text field in characters.
<b>pattern</b>	Pattern supported by <code>java.text.DecimalFormat</code>	A simple pattern which depends on <b>data-type</b> .	Specifies how number is to be parsed and formatted.
<b>language</b>	Lower-case, two-letter codes as defined by ISO-639. Example: "es".	Language of default locale.	Participates in specifying the locale to use.
<b>country</b>	Upper-case, two-letter codes as defined by ISO-3166. Example: "ES".	Country of default locale.	Participates in specifying the locale to use.
<b>variant</b>	Vendor or browser-specific code. Example: "Traditional_WIN".	Variant of default locale.	Participates in specifying the locale to use.
<b>data-type</b>	byte   short   int   long   float   double	double	Base data type of attribute or element value being edited.

`Key, value, ..., key, value` may also specify style parameters [32].

Example:

```
number-field()
number-field(data-type, float,
             pattern, "0.0#####",
             language, en,
             country, "US")
```

## 29. radio-buttons

**radio-buttons**(*key, value, ..., key, value*)

Inserts in generated content a panel containing radio button controls (single selection) or check box controls (multiple selection). These controls can be used to edit the value of the element which is the target of the CSS rule. If "**attribute**, *attribute\_name*" is specified, these controls can be used to edit the value of an attribute of this target element.

Key	Value	Default	Description
<b>attribute</b>	Qualified name of attribute to be edited	No default	Without this parameter, the control is used to edit the value of the element for which the control has been generated.
<b>labels</b>	List of strings separated by new lines (" <code>\A </code> ")	None (use values as labels)	Labels used for the radio buttons or the check boxes. The order of labels must match the order of values.
<b>values</b>	List of strings separated by new lines (" <code>\A </code> ")	None ( <i>dynamically determined by examining the data type of value to be edited</i> )	<p>In single selection mode, clicking on radio button #N sets the element or attribute value being edited to value string #N.</p> <p>In multiple selection mode clicking on check box #N adds/removes value string #N to/from the selected set.</p> <p>The value strings in the selected set are then joined using the character specified by separator ( ' ' by default).</p> <p>The resulting string is assigned to the element or attribute value being edited.</p>
<b>columns</b>	Positive integer	max(10, number of values)	Maximum number of columns used to layout the panel containing the radio buttons or check boxes. Do not specify <b>rows</b> and <b>columns</b> for the same control.
<b>rows</b>	Positive integer	None	Maximum number of rows used to layout the panel containing the radio buttons or check boxes. Do not specify <b>rows</b> and <b>columns</b> for the same control.
<b>selection</b>	single   multiple	single	Specifies single or multiple selection mode.
<b>separator</b>	Single character string	None (values are separated by any type of white space characters)	Character used to join selected value strings in multiple selection mode. The resulting string is assigned to the element or attribute value being edited.

*Key, value, ..., key, value* may also specify style parameters [32].

Examples:

```
radio-buttons(rows, 2)

radio-buttons(attribute, value,
              labels, "Cyan\A Yellow\A Magenta\A Black")

radio-buttons(attribute, value,
              labels, "Cyan\A Yellow\A Magenta\A Black",
              values, "cyan\A yellow\A magenta\A black",
              selection, multiple,
              separator, ",")
```

## 30. remove-attribute-button

**remove-attribute-button**(*attribute, attribute\_name, key, value, ..., key, value*)

Inserts a command-button [36] in generated content which can be used to remove specified attribute.

Optional parameter **check-required** may be set to **yes** (other allowed value is **no**) to specify that no button is to be generated when specified attribute is required.

By default, this button has its icon set to `icon(minus)`.

Example:

```
remove-attribute-button(text, "Remove id",
                        attribute, id,
                        check-required, yes)
```

## 31. replace-button

Inserts a command-button [36] in generated content which can be used to replace the element for which the button has been generated.

Do not specify **command**, **parameter** or **menu** parameters for this type of command-button. A menu of `replace` commands is built dynamically each time this button is clicked.

By default, this button has its icon set to `icon(replace)`.

Example:

```
replace-button()
```

## 32. set-attribute-button

**set-attribute-button**(*attribute*, *attribute\_name*, *key*, *value*, ..., *key*, *value*)

Inserts a command-button [36] in generated content which can be used to give a value to specified attribute. A pop-up menu listing all possible values is displayed when this button is clicked.

This pop-up menu cannot be displayed if the type of the specified attribute is not an enumerated type or is not IDREF or IDREFS. Moreover, when the type of the specified attribute is IDREF or IDREFS, the pop-up menu cannot be displayed if no attributes of type ID have been added to elements in the document.

Optional parameter **unset-attribute** may be set to **yes** (other allowed value is **no**) to specify that a remove attribute command is to be added at the end of the pop-up menu.

By default, this button has its icon set to `icon(pop-down)`.

Example:

```
set-attribute-button(attribute, for,
                    unset-attribute, yes,
                    icon, icon(pop-right));
```

## 33. text

**text**(*text*, *key*, *value*, ..., *key*, *value*)

Inserts in generated content some *styled text*. *Text* specifies the text to be styled. *Key, value, ..., key, value* specify the style parameters [32].

Examples of use:

```
hello:after {
  display: block;
  content: "Hello " text("world", font-weight, bold) "!";
}

map:before {
  display: inline;
  content: paragraph(content(collapser(),
                          " ", element-name()),
```

```

        " ", text(attr(title),
                    font-family, serif,
                    font-size, 1.5em,
                    font-weight, bold)),
border-width, 1px,
border-style, solid,
border-top-color, transparent,
border-left-color, transparent,
border-right-color transparent);
}

```

## 34. text-area

**text-area**(*key, value, ..., key, value*)

Inserts in generated content a (multi-line) text area control. This control can be used to edit the value of the element which is the target of the CSS rule. If "**attribute**, *attribute\_name*" is specified, this control can be used to edit the value of an attribute of this target element.

Key	Value	Default	Description
<b>attribute</b>	Qualified name of attribute to be edited	No default	Without this parameter, the control is used to edit the value of the element for which the control has been generated.
<b>columns</b>	Positive integer	None (the text field expands when the document view is resized)	Width of the control in characters.
<b>rows</b>	Positive integer	3	Number of lines displayed by the control
<b>wrap</b>	none   line   word	none	Specifies how text lines are wrapped.

*Key, value, ..., key, value* may also specify style parameters [32].

Example:

```

text-area(attribute, value,
          columns, 40,
          rows, 2,
          wrap, word)

```

## 35. text-field

**text-field**(*key, value, ..., key, value*)

Inserts in generated content a (single line) text field control. This control can be used to edit the value of the element which is the target of the CSS rule. If "**attribute**, *attribute\_name*" is specified, this control can be used to edit the value of an attribute of this target element.

Key	Value	Default	Description
<b>attribute</b>	Qualified name of attribute to be edited	No default	Without this parameter, the control is used to edit the value of the element for which the control has been generated.
<b>columns</b>	Positive integer	None (the text field expands when the document view is resized)	Width of the control in characters.

*Key, value, ..., key, value* may also specify style parameters [32].

Example:

```
text-field(columns, 10)
```

## 36. time-picker

Similar to date-time-picker [39], except that the dialog box displayed by the button allows to select a time (and not a date/time).

Examples:

```
time30 {
  content: time-picker(attribute, value,
                       format, standard-omit-time-zone);
}

time41 {
  content: time-picker(format, pattern,
                       pattern, "HHmm",
                       language, en,
                       country, "US",
                       columns, 10);
}
```

## 37. value-editor

**value-editor**(*key, value, ..., key, value*)

Inserts a control in generated content. *Which control to insert is found by examining the grammar constraining the document.* This control can be used to edit the value of the element which is the target of the CSS rule. If "**attribute**, *attribute\_name*" is specified, this control can be used to edit the value of an attribute of this target element.

Note that if value-editor is used to generate an editor for an element value and the content type of the target element is not data (XML-Schema examples: `xs:date`, `xs:double`), *no control is generated at all*. A generic style sheet such as `xmldata.css` takes advantage of this feature.

Key	Value	Default	Description
<b>attribute</b>	Qualified name of attribute to be edited	No default	Without this parameter, the control is used to edit the value of the element for which the control has been generated.

*Key, value, ..., key, value* may also specify style parameters [32].

Examples:

```
value-editor()
value-editor(attribute, attribute())
```

## 38. xpath

**xpath**(*XPath\_expression*)

Generalization of standard construct **attr**(*attribute\_name*). Inserts in generated content the value of *XPath\_expression*, an XPath 1.0 expression using the target of the CSS rule (element, comment or processing instruction) at its context node.

Example:

```
xpath("id(@linkend)/@xreflabel")
```

Note that `xpath()`, like `attr()`, is evaluated *once* and this happens when the view of the element is built. This means that in most cases, manually refreshing the view of the element after a change in the document will be needed (use **Select → Redraw (Ctrl+L)**).

Specifying **`attr(foo)`** in a CSS rule implicitly creates a dependency between the value of attribute `foo` and the element which is the target of the CSS rule: the view of the element is automatically rebuilt when the value of its attribute `foo` is changed.

Similarly, specifying **`xpath(whatever)`** in a CSS rule implicitly creates a dependency between the element which is the target of the CSS rule and *all* its attributes: the view of the element is automatically rebuilt when the value of any of its attributes is changed (which too much or not enough depending on the value of the *whatever* XPath expression!).

See also `label()` [51].

## Tip

You are not restricted to the standard functions of XPath 1.0. A few XSLT 1.0 functions such as `document()` are also supported, as well as many very useful extension functions documented in Section 1, “Extension functions” in *XMLmind XML Editor - Commands*.

---

# Chapter 6. Content layouts

## 1. division

**division**(*content*, *key*, *value*, ..., *key*, *value*)

Layout *content* vertically like in a XHTML div.

*Content* is either a single content object such as a string or a list of content objects. In the latter case, special syntax **content**(*content*, ..., *content*) must be used.

*Key*, *value*, ..., *key*, *value* specify optional style parameters [32].

Example:

```
division(content(icon(down), "generated content", icon(up)),
         border-width, 1,
         border-style, solid)
```

## 2. paragraph

**paragraph**(*content*, *key*, *value*, ..., *key*, *value*)

Layout *content* horizontally like in a XHTML p.

*Content* is either a single content object such as a string or a list of content objects. In the latter case, special syntax **content**(*content*, ..., *content*) must be used.

*Key*, *value*, ..., *key*, *value* specify optional style parameters [32].

Example:

```
paragraph(content(icon(right), "generated content", icon(left)),
          border-width, 1,
          border-style, solid)
```

## 3. rows

**rows**(*row\_spec*, ..., *row\_spec*, *key*, *value*, ..., *key*, *value*)

**row**(*cell\_spec*, ..., *cell\_spec*, *key*, *value*, ..., *key*, *value*)

**cell**(*content*, *key*, *value*, ..., *key*, *value*)

Layout *content* in a tabular way like in a XHTML tbody. See also rendering repeating elements as a table [14].

*Content* is either a single content object such as a string or a list of content objects. In the latter case, special syntax **content**(*content*, ..., *content*) must be used.

*Key*, *value*, ..., *key*, *value* specify optional style parameters [32]. Specifying such pairs at the row level is equivalent to specifying them for each cell contained in the row. Specifying such pairs at the rows level allows even more factoring.

Therefore *key*, *value*, ..., *key*, *value* specify optional style parameters [32] for cells *but not for rows and row*. This is different from the behavior of division [59] and paragraph [59] because unlike division and paragraph which are true containers, rows and row are just constructs used to group cells.

Example:

```
row(cell("Category", width, 20ex), cell("Choice #1"),  
    cell("Choice #2"), cell("Choice #3"),  
    font-weight, bold, color, olive,  
    padding-top, 2, padding-right, 2,  
    padding-bottom, 2, padding-left, 2,  
    border-width, 1, border-style, solid);
```



---

# Chapter 7. Display values supported for generated content

This section contain the answer to the following question: given the display of normal content (example: `display: block;`),

- which types of display (example: `display: inline;`),
- which types of content layout (example: `content: paragraph(content(icon(left), "left"));`),

are supported for **:before** and **:after** generated content?

## About *replaced* content

- Replaced content supports all types of content layouts.
- Using generated content for an element having replaced content will give unspecified results.

Content such as `content: icon(left) "middle" attr(foo) circle collapser();` which does not use an explicit layout is said using a *list layout*.

Generated content not described in this section should not be used in XXE.

## 1. display: inline

Displays supported for **:before** and **:after** generated content:

- display: inline. Supported layouts:

- list.

```
b.iil:before,
b.iil:after {
    display: inline;
    content: icon(right) "generated content" icon(left);
    color: navy;
}
```

This paragraph contains ►generated content◀some bold text►generated content◀ of class iil having generated content.

- paragraph.


```
b.iip:before,
b.iip:after {
    display: inline;
    content: paragraph(content(icon(right), "generated content", icon(left)),
                      border-width, 1,
                      border-style, solid);
    color: navy;
}
```

This paragraph contains ►generated content◀some bold text►generated content◀ of class iip having generated content.

- division.



```
b.iid:before,
b.iid:after {
    display: inline;
```

```
content: division(content(icon(down), "generated content", icon(up)),
                  border-width, 1,
                  border-style, solid);
color: navy;
}
```

This paragraph contains  some bold text  of class iid having generated content.

- rows, row or cell (all three give a table).

```
b.iir:before,
b.iir:after {
  display: inline;
  content: row(cell(icon(right)),
               cell("generated content"),
               cell(icon(left)),
               border-width, 1,
               border-style, solid);
  color: navy;
}
```

This paragraph contains  some bold text  of class iir having generated content.

- Other display values are ignored and processed like display: inline.

## 2. display: block

Displays supported for **:before** and **:after** generated content:



- display: inline. Supported layouts:

(The gray frame is used to show that generated content is *inside* the p block.)

- list.

```
p.bil {
  border: 1 solid gray;
  padding: 2;
}

p.bil:before,
p.bil:after {
  display: inline;
  content: icon(right) "generated content" icon(left);
  color: navy;
}
```

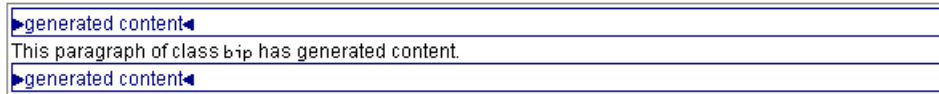
 This paragraph of class bil has generated content. 

- paragraph.

```
p.bip {
  border: 1 solid gray;
  padding: 2;
}

p.bip:before,
p.bip:after {
  display: inline;
  content: paragraph(content(icon(right), "generated content", icon(left)),
                    border-width, 1,
```

```
border-style, solid);  
color: navy;  
}
```



Display: inline, content: paragraph is treated as a special case. The generated paragraph is added before/after normal content but *inside* the whole block. This contrasts with what is done for a generated paragraph with display: block.

- division.

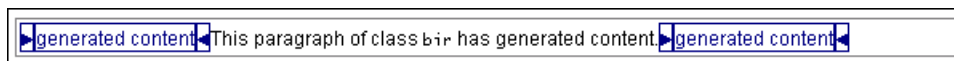
```
p.bid {  
  border: 1 solid gray;  
  padding: 2;  
}  
  
p.bid:before,  
p.bid:after {  
  display: inline;  
  content: division(content(icon(down), "generated content", icon(up)),  
                    border-width, 1,  
                    border-style, solid);  
  
  color: navy;  
}
```



Display: inline, content: division is treated as a special case. The generated division is discarded as a container and all the ``paragraphs" it contains are added before/after normal content but *inside* the whole block. This contrasts with what is done for a generated division with display: block.

- rows, row or cell (all three give a table).

```
p.bir {  
  border: 1 solid gray;  
  padding: 2;  
}  
  
p.bir:before,  
p.bir:after {  
  display: inline;  
  content: row(cell(icon(right)),  
               cell("generated content"),  
               cell(icon(left)),  
               border-width, 1,  
               border-style, solid);  
  
  color: navy;  
}
```



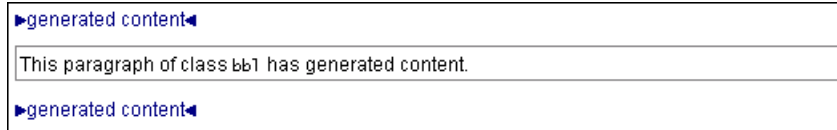
- display: block. Supported layouts:

(The gray frame is used to show that generated content is *outside* the p block.)

- list.

```
p.bbl {
  border: 1 solid gray;
  padding: 2;
}

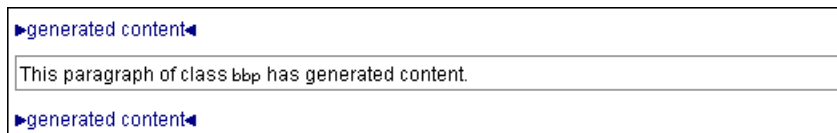
p.bbl:before,
p.bbl:after {
  display: block;
  content: icon(right) "generated content" icon(left);
  color: navy;
  margin-top: 1.33ex;
  margin-bottom: 1.33ex;
}
```



- paragraph.

```
p.bbp {
  border: 1 solid gray;
  padding: 2;
}

p.bbp:before,
p.bbp:after {
  display: block;
  content: paragraph(content(icon(right), "generated content", icon(left)),
                    border-width, 1,
                    border-style, solid);
  color: navy;
  margin-top: 1.33ex;
  margin-bottom: 1.33ex;
}
```

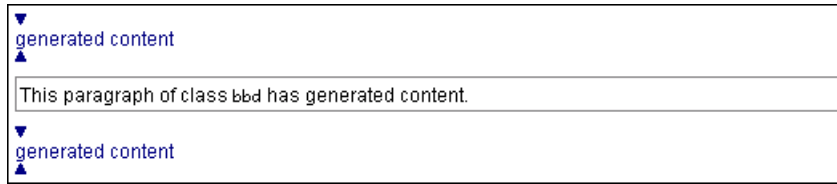


Note that border around generated paragraph is not drawn. It should have been drawn: this is a known deficiency of XXE styling engine. In order to draw this border, move border styles outside `paragraph()`, inside the rule itself.

- division.

```
p.bbd {
  border: 1 solid gray;
  padding: 2;
}

p.bbd:before,
p.bbd:after {
  display: block;
  content: division(content(icon(down), "generated content", icon(up)),
                    border-width, 1,
                    border-style, solid);
  color: navy;
  margin-top: 1.33ex;
  margin-bottom: 1.33ex;
}
```

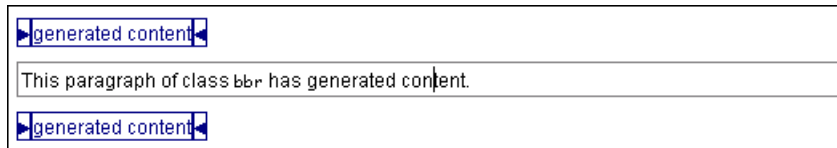


Note that border around generated division is not drawn. It should have been drawn: this is a known deficiency of XXE styling engine. In order to draw this border, move border styles outside `division()`, inside the rule itself.

- rows, row or cell (all three give a table).

```
p.bbr {
  border: 1 solid gray;
  padding: 2;
}

p.bbr:before,
p.bbr:after {
  display: block;
  content: row(cell(icon(right)),
               cell("generated content"),
               cell(icon(left)),
               border-width, 1,
               border-style, solid);
  color: navy;
  margin-top: 1.33ex;
  margin-bottom: 1.33ex;
}
```



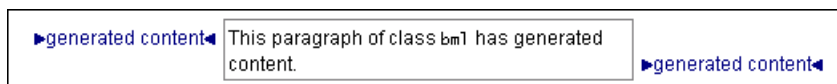
- display: marker. Supported layouts:

(The gray frame is used to show that generated content is *outside* the `p` block.)

- list.

```
p.bml {
  border: 1 solid gray;
  padding: 2;
  margin-left: 20ex;
  margin-right: 20ex;
}

p.bml:before,
p.bml:after {
  display: marker;
  content: icon(right) "generated content" icon(left);
  color: navy;
}
```

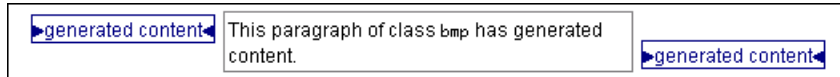


- paragraph.

```
p.bmp {
  border: 1 solid gray;
  padding: 2;
}
```

```
margin-left: 20ex;
margin-right: 20ex;
}

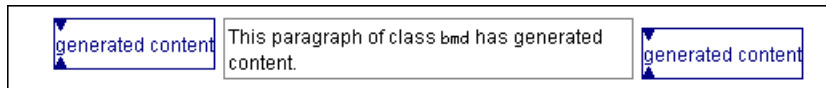
p.bmp:before,
p.bmp:after {
  display: marker;
  content: paragraph(content(icon(right)), "generated content", icon(left)),
           border-width, 1,
           border-style, solid);
  color: navy;
}
```



- division.

```
p.bmd {
  border: 1 solid gray;
  padding: 2;
  margin-left: 20ex;
  margin-right: 20ex;
}

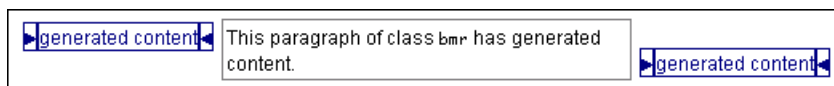
p.bmd:before,
p.bmd:after {
  display: marker;
  content: division(content(icon(down)), "generated content", icon(up)),
           border-width, 1,
           border-style, solid);
  color: navy;
}
```



- rows, row or cell (all three give a table).

```
p.bmr {
  border: 1 solid gray;
  padding: 2;
  margin-left: 20ex;
  margin-right: 20ex;
}

p.bmr:before,
p.bmr:after {
  display: marker;
  content: row(cell(icon(right)),
               cell("generated content"),
               cell(icon(left)),
               border-width, 1,
               border-style, solid);
  color: navy;
}
```



- Other display values are ignored and processed like display: block.

### 3. display: list-item

Display: list-item behaves exactly as display: block [62], except that a content containing the list marker is automatically generated before the list item. Properties list-style-type, list-style-position, list-style-image are used to parametrize the generation of this content.

Example:

```
li {
  display: list-item;
  list-style-type: disc;
}
```

is equivalent to:

```
li {
  display: block;
  margin-left: N; /*make room for the bullet*/
}

li:before {
  display: marker;
  content: disc;
}
```

Note that if the CSS style sheet explicitly specifies a generated content before the list item, display: list-item is strictly equivalent to display: block [62] because, in such case, no content is automatically generated.

### 4. display: table

Displays supported for :before and :after generated content:

- display: block. Same behavior as display: block [62].
- display: marker. Same behavior as display: block [62].
- display: table-row-group or display: table-row. Supported layouts:
  - list.

```
table.tr1:before,
table.tr1:after {
  display: table-row;
  content: icon(right) "generated content" icon(left);
  color: navy;
}
```

▶generated content◀	
Column 1	Column 2
1,1	1,2
2,1	2,2
▶generated content◀	

- paragraph.

```
table.trp:before,
table.trp:after {
  display: table-row;
  content: paragraph(content(icon(right), "generated content", icon(left)),
                    border-width, 1,
                    border-style, solid);
}
```

```
color: navy;
}
```

Table of class trp

generated content	
Column 1	Column 2
1,1	1,2
2,1	2,2
generated content	

- division

```
table.trd:before,
table.trd:after {
  display: table-row;
  content: division(content(icon(down), "generated content", icon(up)),
                    border-width, 1,
                    border-style, solid);
  color: navy;
}
```

Table of class trd

generated content	
Column 1	Column 2
1,1	1,2
2,1	2,2
generated content	

- rows, row or cell (all three give one or several rows).

```
table.trr:before,
table.trr:after {
  display: table-row;
  content: row(cell(icon(right)),
               cell("generated content"),
               cell(icon(left)),
               border-width, 1,
               border-style, solid);
  color: navy;
}
```

Table of class trr

	generated content
Column 1	Column 2
1,1	1,2
2,1	2,2
	generated content

Note that generated row has been merged to normal content. See also rendering repeating elements as a table [14].

- Other display values are ignored and processed like display: block.

## 5. display: table-row-group

Displays supported for **:before** and **:after** generated content:

- display: table-row. Supported layouts:



- list.

```
thead.grl:before,  
thead.grl:after {  
    display: table-row;  
    content: icon(right) "generated content" icon(left);  
    color: navy;  
}
```

generated content	
Column 1	Column 2
generated content	
1,1	1,2
2,1	2,2

- paragraph.

```
thead.grp:before,  
thead.grp:after {  
    display: table-row;  
    content: paragraph(content(icon(right), "generated content", icon(left)),  
                        border-width, 1,  
                        border-style, solid);  
    color: navy;  
}
```

generated content	
Column 1	Column 2
generated content	
1,1	1,2
2,1	2,2

- division

```
thead.grd:before,  
thead.grd:after {  
    display: table-row;  
    content: division(content(icon(down), "generated content", icon(up)),  
                      border-width, 1,  
                      border-style, solid);  
    color: navy;  
}
```

generated content	
Column 1	Column 2
generated content	
1,1	1,2
2,1	2,2

- rows, row or cell (all three give one or several rows).

```
thead.grr:before,  
thead.grr:after {  
    display: table-row;  
    content: row(cell(icon(right)),  
                 cell("generated content"),  
                 cell(icon(left)),  
                 border-width, 1,  
                 border-style, solid);  
    color: navy;  
}
```

generated content	
Column 1	Column 2
generated content	
1,1	1,2
2,1	2,2

- Other display values are ignored and processed like display: table-row.

## 6. display: table-row

Displays supported for :before and :after generated content:

- display: table-cell. Supported layouts:

- list.

```
tr.rcl:before,
tr.rcl:after {
  display: table-cell;
  content: icon(right) "generated content" icon(left);
  color: navy;
}
```

Column 1	Column 2	
generated content	1,1	1,2 generated content
2,1	2,2	

- paragraph.

```
tr.rcp:before,
tr.rcp:after {
  display: table-cell;
  content: paragraph(content(icon(right), "generated content", icon(left)),
    border-width, 1,
    border-style, solid);
  color: navy;
}
```

Column 1	Column 2	
generated content	1,1	1,2 generated content
2,1	2,2	

- division

```
tr.rcd:before,
tr.rcd:after {
  display: table-cell;
  content: division(content(icon(down), "generated content", icon(up)),
    border-width, 1,
    border-style, solid);
  color: navy;
}
```

Column 1	Column 2	
generated content	1,1	1,2 generated content
2,1	2,2	

- rows, row or cell (all three give a table).

```
tr.rcr:before,  
tr.rcr:after {  
  display: table-cell;  
  content: row(cell(icon(right)),  
               cell("generated content"),  
               cell(icon(left)),  
               border-width, 1,  
               border-style, solid);  
  color: navy;  
}
```

Column 1	Column 2	
generated content	1,1	1,2 generated content
2,1	2,2	

- Other display values are ignored and processed like display: table-cell.

## 7. display: table-cell

Same behavior as display: block [62].