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

Hussein Shafie
XMLmind Software
<xmleditor-support@xmlmind.com>

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

Hussein Shafie
XMLmind Software
<xmleditor-support@xmlmind.com>

Publication date March 22, 2024

Abstract

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

Table of Contents

List of Figures

List of Tables

Part I. Guide

Table of Contents

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() [74] or the image-viewport() [75] content objects.)

Chapter 2. List of supported CSS properties

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	[*`background-color`* || *`background-image`* || *`background-repeat`* || *`background-attachment`* || *`background-position`*] | inherit	Note that property background-attachment is not supported at all. Property background-attachment is always assumed to be `scroll`.
background-color	*color* | transparent | inherit | normal[a]	-

background-image	*`url`* | *`icon`* [71] | linear-gradient() | none | inherit	Built-in images such as `icon(hatch1, #FFF0F0)` are supported in addition to something like `url(backgrounds/gradient23.png)`.
		
		`linear-gradient()` is a CSS3 image value only supported as a `background-image`.
		
		Moreover, color stops expressed as pixels or as percentages outside the [0,100] range are not supported. For example:
		
		`linear-gradient(red, white 20%, blue)`
		
		is supported, while
		
		`linear-gradient(red -50px, white, blue)`
		
		`linear-gradient(red, white -50%, black 150%, blue)`

Property	Value	Restrictions
		are not.
background-position	0% 0% | top left | left top | *length*{1,2} | inherit	Percentages other than `0% 0%` and positions other than `top left` and `left top` are correctly parsed but generally, not correctly rendered.
background-repeat	repeat | repeat-x | repeat-y | no-repeat | inherit	-
border	[*border-width* || *border-style* || *border-color*] | inherit	-
border-color	*side_value*{1,4}	-
border-bottom-color	*color* | transparent | inherit	-
border-left-color	"	-
border-right-color	"	-
border-top-color	"	-
border-top-left-radius	*length* | *percentage* [*length* | *percentage*]? | inherit	-  All the following border-radius properties are CSS3, and not CSS2, properties.
border-top-right-radius	"	-
border-bottom-left-radius	"	-
border-bottom-right-radius	"	-
border-radius	[*length* | *percentage*]{1,4} [/ [*length* | *percentage*]{1,4}]?	-
border-style	*side_value*{1,4}	-
border-top-style	none | hidden | dotted | dashed | solid | double | groove | ridge | inset | outset	-
border-right-style	"	-
border-bottom-style	"	-
border-left-style	"	-
border-width	thin | thick | medium | *length* | inherit	Note that border-top-width, border-right-width, border-bottom-width and border-left-width,

Property	Value	Restrictions
		while successfully parsed are not really supported.
color	*color* | inherit | normal[a]	System colors (`ButtonFace`, `ButtonText`, etc) are supported by all color properties (`color`, `background-color`, `border-color`, etc).
counter-reset, counter-increment	[*`identifier integer`*?]+ | 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	`table-header-group` and `table-footer-group` are considered to be equivalent to `table-row-group`. Normally `table-header-groups` should be displayed before first `table-row-group` and `table-footer-groups` should be displayed after last `table-row-group`.
font	[[*`font-style`* || *`font-variant`* || *`font-weight`*]? *`font-size`* [/ *`line-height`*]? *`font-family`*] | inherit	Font-variant is correctly parsed, but not honored. System fonts (`caption`, `message-box`, etc) are supported too.
font-family	[[*name* | *generic*] ,]* [*name* | *generic*] | inherit | normal[a]	The generic font families `cursive` and `fantasy` are not supported.
font-size	medium | small | large | x-small | x-large | xx-small | xx-large | smaller | larger | *length* | *percentage* | inherit	-
font-style	normal | italic | oblique | inherit	Italic and oblique are aliases.
font-weight	normal | bold | bolder| lighter | inherit	No *N*00. Bolder and bold are aliases. Lighter and normal are aliases.
line-height	normal | *`number`* | *`length`* | *`percentage`* | inherit	-
list-style-image	*`URI`* | none | | inherit	Also supports icon(*`name`*).
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	[*`list-style-type`* || *`list-style-position`* || *`list-style-image`*] | inherit	-
margin	*side_value*{1,4}	-

Property	Value	Restrictions
margin-bottom	*length* | auto | inherit	No *percentage*.
margin-left	"	-
margin-right	"	-
margin-top	"	-
padding	*side_value*{1,4}	-
padding-bottom	*length* | inherit	No *percentage*.
padding-left	"	-
padding-right	"	-
padding-top	"	-
text-align	left | right | center | justify | inherit	Justify and left are aliases.
text-decoration	none | underline | overline | line-through | inherit	No blink.
text-indent	*length* | inherit	No *percentage*.
vertical-align	baseline | middle | sub | super | text-top | top | text-bottom | bottom | inherit	No *length*, *percentage*.
white-space	normal | pre | nowrap | inherit	-

[a] "`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	*length length*?	-
caption-side	top | bottom	left, right, inherit are not supported.
content	normal | none | *string* | *uri* | attr(*X*) | open-quote | close-quote | no-open-quote | no-close-quote | counter(*name*) | counter(*name*, *style*) | counters(*name*, *separ*) | counters(*name*, *separ*, *style*) | 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	*length* | auto	No *percentage*. This property is currently ignored.
marker-offset	*length* | auto | fill	No *percentage*.
width	*length* | auto	No *percentage*. This property is currently only useful to specify the minimum width of a table cell.

Implementation specificities and 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;`. See Section 8, “New values for the display property” [21].

- 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`.

- When the directionality of the element (e.g. the element has or inherits attribute `dir="rtl"`) which is the subject of the CSS rule is right-to-left and when its display is `list-item`, `block`, `inline-block` or `inline`.
  - `text-align: left;` is processed as if it were `text-align: start;` and `text-align: right;` is processed as if it were `text-align: end;`
  - margin-left is processed as if it were *CSS logical property* margin-inline-start. margin-right is processed as if it were CSS logical property margin-inline-end.
  - padding-left is processed as if it were CSS logical property padding-inline-start. padding-right is processed as if it were CSS logical property padding-inline-end.
  - border-left-color is processed as if it were CSS logical property border-inline-start-color. border-right-color is processed as if it were CSS logical property border-inline-end-color.
  - border-left-style is processed as if it were CSS logical property border-inline-start-style. border-right-style is processed as if it were CSS logical property border-inline-end-style.
  - Properties border-top-left-radius, border-bottom-left-radius, border-top-right-radius, border-bottom-right-radius are processed as if they were CSS logical properties border-start-start-radius, border-start-end-radius, border-end-start-radius, border-end-end-radius respectively.

- `:first-letter` and `:first-line` pseudo-elements are ignored.

- The `!important` specifier is ignored.

- `@import` rules may be specified anywhere in a CSS file. This is not allowed by the CSS standard which requires such rules to precede all other rules except the `@charset` rule.

- The following properties are correctly parsed but generally, not correctly rendered:
  - border-top, border-right, border-bottom, border-left.
  - border-top-style, border-right-style, border-bottom-style, border-left-style.
  - border-top-width, border-right-width, border-bottom-width, border-left-width.
  - background-position: *`percentage`* other that 0% 0% and keywords other than `left top` and `top left`.

- Property `border-collapse` is parsed but not implemented. Note that only `border-collapse:separate` is supported.

- `@font-face` rules are supported however the required `src` property must contain at least one `url()` item ending with "`.ttf`" or "`.otf`". This feature is mainly useful to be able to use icons fonts such as "Font Awesome". Example:

```
@font-face {
    font-family: icon;
    src: url(https://maxcdn.bootstrapcdn.com/font-awesome/¬
4.4.0/fonts/fontawesome-webfont.ttf);
}
```

```
quote:before {
    content: "\f10d"; /*fa-quote-left*/
    font-family: icon;
    color: gray;
}

quote:after {
    content: "\f10e"; /*fa-quote-right*/
    font-family: icon;
    color: gray;
}
```

Chapter 3. Extensions related to generated content

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 [52].

  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 [93].

  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 [95].

  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

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.

```
/*
 * Copyright (c) 2017-2024 XMLmind Software. All rights reserved.
 *
 * This file is part of the XMLmind XML Editor project.
 * For conditions of distribution and use, see the accompanying legal.txt file.
 *
 * Built-in CSS rules.
 */

/* ------------------------------------------------------------------------
   Comments and processing-instructions
   ------------------------------------------------------------------------ */

*::comment,
*::processing-instruction {
    display: inherit;
    white-space: pre;
    text-align: left;
    font-family: monospace;
    font-style: normal;
    font-weight: normal;
    font-size: small;
    margin: 0.25ex;
}

*::comment {
    background-color: #FFFFDD;
    color: #606000;
}

*::processing-instruction {
    background-color: #DDFFDD;
```

```
    color: #006000;
}

/* ------------------------------------------------------------------------
   Remark
   ------------------------------------------------------------------------ */

*::processing-instruction(xxe-remark) {
    display: inline;
    content: gadget("com.xmlmind.xmleditapp.cmd.diff.RemarkIndicator");
    font-size: smaller;
    background-color: transparent;
}

/* ------------------------------------------------------------------------
   Nodes marked read-only
   ------------------------------------------------------------------------ */

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

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

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

/* ------------------------------------------------------------------------
   XInclude directive
   ------------------------------------------------------------------------ */

@namespace xi url(http://www.w3.org/2001/XInclude);

@property-value xi-include-content()
    paragraph(content(icon(pop-se, 0), " ", element-name(),
                      " href=", attr(href),
                      " xpointer=", xpath("if(@xpointer, @xpointer, @fragid)")));

/* The xpointer attribute must not be present when parse="text" */

@property-value xi-text-include-content()
    paragraph(content(icon(pop-se, 0), " ", element-name(),
                       " href=", attr(href),
                       " fragid=", attr(fragid),
                       " parse=", attr(parse),
                       " encoding=", attr(encoding)));

xi|include {
    display: block;
    margin-bottom: 1.33ex;
}
```

```
xi|include:before {
    display: block;
    content: xi-include-content();
    font-style: italic;
    color: gray;
    margin-top: 1.33ex;
}

xi|include[parse*="text"]:before {
    content: xi-text-include-content();
}

xi|include:empty {
    display: inherit;
    content: xi-include-content();
    font-style: italic;
    color: gray;
}

xi|include:empty:before {
    content: "";
}

xi|include[parse=text]:empty {
    content: xi-text-include-content();
}

xi|include[parse=text]:empty:before {
    content: "";
}

xi|fallback {
    display: block;
    font-family: normal;
    font-size: medium;
    font-weight: normal;
    font-style: normal;
    background-color: normal;
    color: normal;
    padding: 0.5ex;
    border: 1px dotted gray;
    margin: 0.5ex;
}

/* ----------------------------------------------------------------------
   When printing
   ---------------------------------------------------------------------- */

@media print {
    *::comment,
    *::processing-instruction {
```

```
        display: none;
    }

    *:property(prop|readOnly) {
        background-color: transparent;
    }
}
```

In practice, this just means that you have nothing special to do to style nodes marked as being read-only, XIncludes, comments, processing instructions.

2. CSS3 selectors

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

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

3. Styling an element which contains a specific processing instruction

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

Several processing instruction targets may be specified, in which case, the :contains-processing-instruction() test will pass if the subject of the test directly contains a processing instruction having any of the specified targets.

Example: display all XHTML spans containing one or more marker processing instruction with a yellow background.

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

4. Styling an element which contains a specific child element

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

Several child element names may be specified, in which case, the `:contains-element()` test will pass if the subject of the test directly contains a child element having any of the specified names.

Note that:

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

is very different from:

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

In the first case, the subject 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) {❶
    color: red;
}

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

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

p:contains-element(foo|hl) {❸
    color: blue;
}

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

p:contains-element(hl) {❹
    color: yellow;
}

*|*:contains-element(*|hl) {❺
    text-decoration: underline;
}

*|hl {
```

```
    display: inline;
}
```

❶ Element with local name `p`, whatever is its namespace, containing a `i` whatever is its namespace, gets a red color.

❷ Element with local name `p`, whatever is its namespace, containing a `{}b`, gets a green color.

❸ Element with local name `p`, whatever is its namespace, containing a `{http://foo.com}hl`, gets a blue color.

❹ Element `{http://bar.com}p`, containing a `{http://bar.com}hl`, gets a yellow color.

❺ Any element having a child with local name `hl`, whatever is the namespace of this `hl`, 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
element-name()	The fully qualified name of the element.	ns:myElement-1
element-local-name()	Local part of element name.	myElement-1
element-namespace-uri()	Namespace URI of element name.	http://acme.com/ns/foo/bar
element-label()	Local part of element name, made more readable.	My element 1

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. Note that pseudo-functions such as `attr()` or `xpath()` [90] are evaluated in order to obtain their string values.

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()` [74]):

```
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 (DocBook):

```
@extension "com.xmlmind.xmleditext.docbook.TableSupport true";
```

In the above example, `com.xmlmind.xmleditapp.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 an other extension class may be used to localize generated content. If two `@extension`s 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 7, *All stylesheet extension points*.

- 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 public constructor with the following signature: *`Constructor`*`(java.lang.String[] parameters, com.xmlmind.xmledit.styled.ViewFactoryBase factory)`.

- Invoking the constructor of the extension class may have side effects such as registering intrinsic style specifications (`com.xmlmind.xmledit.stylesheet.StyleSpecs`, see Chapter 7, *All stylesheet extension points*) with the `com.xmlmind.xmledit.styled.ViewFactoryBase` 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 7, *All stylesheet extension points*):

```
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 7, *All stylesheet extension points*): `com.xmlmind.xmledit.stylesheet.StyleValue` *`Method`*`(com.xmlmind.xmledit.stylesheet.StyleValue[] parameters, com.xmlmind.xmledit.doc.Node contextNode, com.xmlmind.xmledit.styled.ViewFactoryBase 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`.

7.2.1. Invoking a static extension method

Alternatively `invoke` may be used to invoke a *static* extension method by specifying its fully qualified name. DocBook example:

```
link {
    tooltip-content: attr(linkend)
      "\A \A "
      invoke("com.xmlmind.xmleditapp.dndview.ExtensionMethods.bindingDescription",
             "docb.followLinkAt");
}
```

In the above example, the static method is called `bindingDescription` and it is found in “ordinary class”[1] `com.xmlmind.xmleditapp.dndview.ExtensionMethods`.

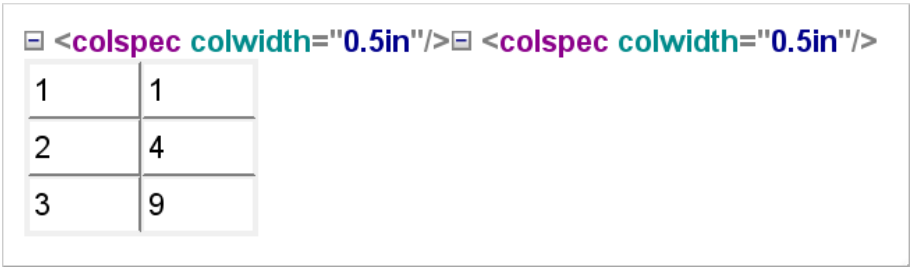
8. New values for the display property

`display: tree`

May be used to mix styled views and tree 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.

[1]That is, unlike an extension class, this class will not be instantiated by the CSS engine and it is not required to have a public constructor having the following signature: *`Constructor`*`(java.lang.String[] parameters, com.xmlmind.xmledit.styled.ViewFactoryBase factory)`.

Figure 4.1. A DocBook table in which `colspec` elements have `display: tree`



`display: view(`*`class_name`*`)`

`Class_name` is the fully qualified name of a Java™ class implementing `com.xmlmind.xmledit.styledview.StyledElementViewFactory`. A `StyledElementViewFactory` is used to create custom styled views for some elements. Such factory classes are used to implement the styled view of the XHTML `ruby` element and the styled views of most MathML elements. More information in Section 7, "`StyledElementViewFactory`".

9. Rendering repeating elements as a table

In order to style repeating elements as a table, you'll generally 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).

If you don't have child elements acting as table rows, then you need to specify extension property **column-count** on the element styled as a table. See Section 9.1, "Anonymous rows" [24].

- 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.

  For example, the above properties could be used to style XHTML-like tables as follows:

```
table {
    display: table;
}

tr {
    display: table-row;
}

td {
    display: table-cell;
}

td[colspan] {
    column-span: concatenate(attr(colspan));
}

td[rowspan] {
    row-span: concatenate(attr(rowspan));
}
```

Notice the use of pseudo-function `concatenate()` [19] to parse the value of an attribute as a number.

- 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

9.1. Anonymous rows

If you don't have child elements acting as table rows, then you need to specify extension property **column-count** on the element styled as a table. The value of this property is a strictly positive integer number. This property has no initial value because by default, tables are expected to have child elements acting as rows.

Tables for which anonymous rows are automatically generated cannot have child elements with `display:table-row` and the **row-span** extension property (see above [22]) is not supported for grandchild elements acting as table cells. Other than the two aforementioned limitations, such tables support all the features of normal tables (borders around the table and the cells, **column-span**, generated content, etc).

Note that in the case of tables for which anonymous rows are automatically generated, you'll often have to explicitly specify standard property **width** (length or percentage) and extension property **start-column** (positive integer; see above [23]) on table cells in order to achieve the kind of layout you want.

XHTML `dl` (definition list) example:

```
dl {
    display: table;
    column-count: 2;
}

dt,
dd {
    display: table-cell;
    padding: 0.5ex;
}

dt {
    width: 25%;
    start-column: 0;
}

dd {
    width: 50%;
    start-column: 1;
}
```

XHTML source:

```
<dl>
  <dt>Term #1</dt>
  <dd>Definition of term #1.</dd>
  <dt>Term #2a</dt>
  <dt>Term #2b</dt>
  <dd>Definition of term #2.<p>More info about term #2.</p></dd>
  <dt>Term #3</dt>
  <dd>Definition of term #3.</dd>
</dl>
```

Rendered as:

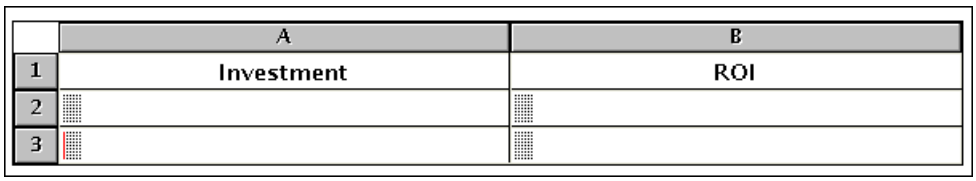
Term #1	Definition of term #1.
Term #2a	
Term #2b	Definition of term #2.
	More info about term #2.
Term #3	Definition of term #3.

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;
}
```



11. Collapsible blocks and tables

Elements with `display: block`, `inline-block`, `table`, `inline-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	yes block or table can be collapsed and expanded no block or table cannot be collapsed and expanded
collapsed	yes | no	no	yes block or table is initially collapsed no block or table is initially expanded
not-collapsible-head	*non-negative integer*	0	Number of child elements found at the beginning of the block or table which must be kept visible even if the block or table is collapsed. If the block or table has fewer child elements than those specified proprerty not-collapsible-head or in property not-collapsible-foot, then the block or table is *not* considered to be collapsible.
not-collapsible-foot	*non-negative integer*	0	Number of child elements found at the beginning at the end of the block or table which must be kept visible even if the block or table is collapsed.
collapsed-content	*same as property content*	*no content*	Content which must be displayed (in lieu of hidden graphical items) when the block or table is collapsed.

Property	Value	Initial value	Description
			Note that this content is transformed to an image before being used. Therefore this type of generated content cannot wrap at word boundaries. Do not specify label() [78] as a value for `collapsed-content`. Instead specify xpath() [90].
collapsed-content-align	`auto|left|center|right`	auto	Specifies how the `collapsed-content` image is to be horizontally aligned. Special value `auto` means that the `collapsed-content` image must be horizontally aligned just like the normal content it represents.

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 [56].

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();
}
```

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**).

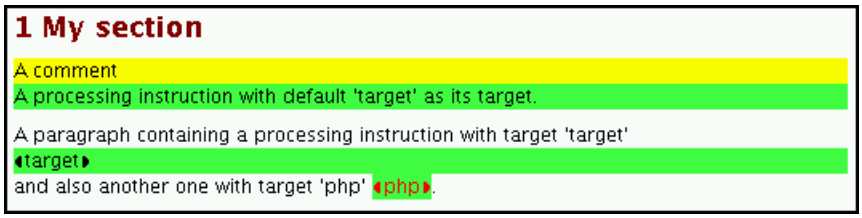
The replaced content of a processing instruction may contain `processing-instruction-target()` which is replaced by the target of the processing instruction subject of the rule.

Similarly the replaced content of a comment or a processing instruction may contain `text-content()` which is replaced by the text of the text node 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:



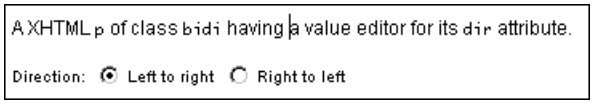
Note that it is also possible to use CSS3-like syntax `::comment` and `::processing-instruction` instead of CSS2-like syntax `:comment` and `:processing-instruction`.

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 [84] are used to set the `dir` attribute of a `p` of class `bidi`.

```
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 {❶
    display: block;
    content: attributes();
}

p.bidi2::attribute(dir) {❷
    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;
}
```

❶ First rule inserts an `attributes()` container [54] 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.

❷ 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.

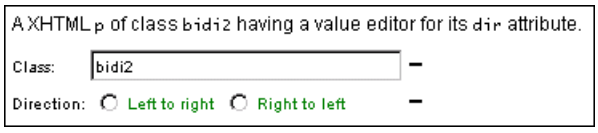
Table 4.2. Properties used to specify generated content for attributes

Property	Value	Initial value	Description
attribute-content-left	Any value allowed for the **content:** property plus `attribute-*()` pseudo functions (see below [33]).	"" (no content)	Generated content for the attribute which is the target of the `:attribute()` rule that goes to the left column of the `attributes()` container.
attribute-content-middle	Any value allowed for the **content:** property plus `attribute-*()` pseudo functions (see below [33]).	"" (no content)	Generated content for the attribute which is the target of the `:attribute()` rule that goes to the middle column of the `attributes()` container.
attribute-content-right	Any value allowed for the **content:** property plus `attribute-*()` pseudo functions (see below [33]).	"" (no content)	Generated content for the attribute which is the target of the `:attribute()` rule that goes to the right column of the `attributes()` container.
show-attribute	**never** | **always** | **when-added**	**when-added**	never Never display this attribute in the `attributes()` container. always Always display this attribute in the `attributes()` container even if the attribute has not yet

Property	Value	Initial value	Description
			been added to the element. when-added Display this attribute in the `attributes()` container if the attribute has been added to the element.

Same example as above with all attributes a `p` of class `bidi2`, 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() {❶
    attribute-content-left: attribute-label() ":";❷
    attribute-content-middle: ❸value-editor(attribute, attribute());❹
    attribute-content-right: remove-attribute-button(attribute, attribute());❺
    show-attribute: when-added;
    font-size: smaller;
}

p.bidi2::attribute(dir) {❻
    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;
}
```



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 `bidi2` has no `dir` attribute yet.

By default (this can be specified [53]):

- 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.

❶ This rule specifies the generated content for all attributes of a `p` of class `bidi2`.

❷ `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

❸ `value-editor()` [90] will automatically find a suitable value editor based on the data type of attribute which is the target of the rule.

❹ `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.

❺ See `remove-attribute-button()` [85].

❻ 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, XXE).

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"

Pattern	Meaning
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 sub-string "o".

An example of application property is `{http://www.xmlmind.com/xmleditor/namespace/property}sourceURL`, the location of the file from which an XML node has been loaded. Excerpt of *`XXE_install_dir`*`/addon/config/common/css/visible_inclusions.imp`:

```
@namespace prop "http://www.xmlmind.com/xmleditor/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(docbook-config:css/docbook.css);❶
      .
      .
   my customization here
      .
```

❶ Note that `@import "docbook-config: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. Weak @import

The "`---`" prefix before an URL instructs XXE to silently skip the `@import` when the URL cannot be successfully resolved. Example:

```
/* Do not report an error if the "MathML support" add-on
   has not been installed. */
@import url(---mathml-config:xhtml5/mathml_support.imp);
```

17. Translating messages in the content generated by a CSS style sheet

This is done as follows:

1. All the messages to be translated must be enclosed in the `_()` pseudo-function. Example:

   ```
   .hello:before {
       content: _("Hello world!");
   }
   ```

2. An `@translation` at-rule pointing to a Java® property file must be added at the top of the CSS stylesheet. Example:

   ```
   @translation url(hello_en.properties);
   ```

   where file `hello_en.properties` contains:

   ```
   sayHello=Hello world!
   ```

   The name of the Java property file must end with "*`_ll`*`.properties`", where "*`ll`*" is a lower-case two-letter ISO 639 language code.

   This language code specifies the language of the messages contained in the Java property file. This language is English (code "`en`") in the case of the above example. Note that country variants (`en-US`, `en-GB`, `fr-FR`, `fr-CA`, etc) are not supported here.

A relative URL is resolved against the URL of the CSS style sheet. In the example above, file `hello_en.properties` is expected to be found in the same directory as the CSS style sheet “including” it.

3. An additional Java property file must be created for each available language.

   French example: `hello_fr.properties`:

   ```
   sayHello=Bonjour monde!
   ```

   German example: `hello_de.properties`:

   ```
   sayHello=Hallo Welt!
   ```

   These additional property files must use exactly the same *property keys* as those found in the *reference property file*.

   The reference property file is the file included in the CSS style sheet by the means of the `@translation` at-rule.

   In the example above, the reference property file is `hello_en.properties`. The author of this file has arbitrarily chosen to use property key "`sayHello`" to identify message "`Hello world!`" .

   These additional property files may be found in the same directory as the reference property file or, alternatively, they may be top level items contained in a `JAR` file referenced in the `CLASSPATH`. Alternate location example:

   ```
   $ jar tvf fr_translation.jar
   ...
   hello_fr.properties
   ...
   com/xmlmind/netutil/Messages_fr.properties
   ```

   This alternate location is the one used by the "*Translate XMLmind XML Editor*" add-on.

Notes:

- A Java property file automatically trims whitespace found before and after a message. That's why leading and trailing whitespace are not considered when searching the translation corresponding to a given message.

  Example:

  ```
  .hello2:before {
      content: url(left.png) _(" Hello world! ") url(right.png);
  }
  ```

  In the example above, the author of the style sheets wants to have a space character before and after "`Hello world!`". However in the `_ll.properties` message files, there is no need to add these leading and trailing space characters.

  Reference English message file `hello_en.properties`:

```
sayHello=Hello world!
```

French message file `hello_fr.properties`:

```
sayHello=Bonjour monde!
```

Despite the fact that there is no space character before and after "`Bonjour monde!`", when using a French locale, the CSS style sheet is equivalent to:

```
.hello2:before {
    content: url(left.png) " Bonjour monde! " url(right.png);
}
```

- In some cases, you'll want to translate strings found inside an XPath expression [90]. Example:

```
.hello3:before {
    content: xpath("concat(../@title, ' should contain ', 'Hello world!')");
}
```

This is done as follows:

```
.hello3:before {
    content: xpath(_("concat(../@title, '%0', '%1')", " should contain ", "Hello world!"));
}
```

with message file `hello_en.properties` containing:

```
sayHello=Bonjour monde!

requirement=should contain
```

When the `_()` pseudo-function is passed more than one message:

1. All the message arguments are translated.
2. The first message translation is considered to be a *template* containing up to 10 variables: `%0`, `%1`, ..., `%9`.
3. The message translations other than the first one are considered to be the values of variables `%0`, `%1`, ..., `%9`.
4. The variables are substituted with their values in the template. The result of this substitution is the string passed to pseudo-function `xpath()`.

This facility is mainly useful to cope with messages found inside XPath expressions, but you can also use it anywhere it helps. Example:

```
.hello4:before {
    content: _('Greetings %0 "%1"', " should contain ", "Hello world!");
}
```

18. 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.

18.1. @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;
}
```

18.2. @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, #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)));
}
```

19. 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;
}
```

20. 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;
}
```

21. 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`).

22. Specifying the type of counter created by list-style-type

A CSS rule such as:

```
ol[type="I"] {
    list-style-type: upper-roman;
}
```

implicitly creates a counter, but does not allow to give it a name or to parameterize it in any way.

Extension property list-item-counter may be used to specify the counter implicitly created by list-style-type. Example:

```
ol > li {
    list-item-counter: counter(list-item) ". ";
    counter-increment: list-item;
}
```

- This property applies to elements having a `list-item` display.
- This property is not inherited and cannot be inherited by specifying keyword `inherit`.
- This value of property list-item-counter is similar to the value of property content, though it will almost always contain `counter()`, `counters()`, `simple-counter() [45]` or `simple-counters() [45]`.
- The default value of list-item-counter is: `simple-counter(n, format)`, where *`format`* is implicitly specified by the value of property list-style-type.

23. When showing tags, hide tags for some elements

Menu **View** → **Show Tags** in *XMLmind XML Editor - Online Help* allows to display element tags in a styled view. But what if you don't want this to happen for some elements?

Extension property hide-tags allows to specify this. MathML example:

```
mml|semantics,
mml|annotation,
mml|annotation-xml {
    display: block;
    hide-tags: yes;
    margin-top: 1.33ex;
    margin-bottom: 1.33ex;
}
```

The value of this property may be `yes` or `no`. Initially, its value is `no`.

This property is not inherited and cannot be inherited by specifying keyword `inherit`.

24. Specifying the tooltip to be displayed for a given element.

Extension property tooltip-content allows to specify the textual contents of a *tooltip* (also called "balloon help" or "mouse hover"). This tooltip is displayed by **XXE** when the user moves the mouse over the view of the element which is the subject of the CSS rule containing tooltip-content.

The value of this property is any generated content which evaluates as text. A generated content which does not evaluate as text is simply ignored.

This property is not inherited and cannot be inherited by specifying keyword `inherit`.

Examples:

```
pre.tooltip {
    tooltip-content: "<html>❶<body><tt>pre=<big>" element-local-name()¬
"</big></tt><br><small>(HTML tooltip)</small>";
}

pre.tooltip b {
    tooltip-content: "b/@lang=" attr(lang);
}

pre.tooltip i {
    tooltip-content: "i inside \A "❷ xpath("local-name(parent::*)");
}

pre.tooltip u {
    tooltip-content: "u is not a " icon(left-link)❸ "link";
}
```

❶ A value which starts with "`<html>`" is parsed as HTML 3.2 and allows to specify a tooltip containing “rich text”.

❷ A value may contain newline characters ("`\A `").

❸ Generated content `icon(left-link)`, which evaluates as an image, is silently skipped.

25. Adding a *decoded URI* as generated content

The `decode-uri()` extension pseudo-function “%-decodes” its argument. This argument is assumed to be the string representation of a possibly “%-encoded” URI. For example, `decode-uri()` can be used to display "`D%C3%A9j%C3%A0%C2%20vu.dita`", the value of an `href` attribute, as "`Déjà vu.dita`".

DITA example:

```
xref[href]:after {
    content: icon(right-link, 0) decode-uri(attr(href));
}
```

Part II. Reference

Table of Contents

Chapter 5. Content objects

- *XXE_install_dir*/demo/forms/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/forms/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)
```

  The default font and colors of content objects such as `check-box`, `combo-box`, etc

  - The value of CSS property `font` is inherited from the style of the element for which the content object is generated.

  - The value of CSS property `background-color` is *not* inherited. A content object has an intrinsic background color which comes from the Java™ “Look & Feel” used for the XML editor.

  - If a solid `background-color` has been explicitly specified for the content object or if the content object is intrinsically transparent (`radio-buttons` [84], `check-box` [55]), then the value of CSS property `color` is inherited from the style of the element for which the content object is generated.

    Otherwise, the content object has an intrinsic foreground color which comes from the Java “Look & Feel” used for the XML editor.

- *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 is 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(foreground))
```

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

- All pseudo-functions generating controls (text-field [89], list [80], etc) also support the following color specifications:

Key	Value	Default	Description
missing-color	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.
missing-background-color	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.
error-color	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.
error-background-color	Color	None (no special background color)	Background 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.

Example:

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

- All pseudo-functions generating content (except icon() [71] and xpath() [90]) 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 [69].

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

1. add-attribute-button

Inserts a command-button [58] 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
check-has-attributes	Boolean: yes|no, 1|0, true|false, on|off	no	**yes** means: do not generate this button when the target element has no attributes (attribute wildcards and `xsi`:* attributes are ignored).
use-default-attribute-value	Boolean: yes|no, 1|0, true|false, on|off	no	**yes** 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 [30].

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
wrap-rows	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 [52].

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
attribute	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.
label	String	None	Label used for the check box.
unchecked-value	String	None	In normal mode, unchecking the control assigns this value to the attribute or element value being edited.
checked-value	String	None	In normal mode, checking the control assigns this value to the attribute or element value being edited.
allow-empty-checked-value	Boolean: yes|no, 1|0, "true"|"false", "on"|"off"	no	Consider the empty string ("") to be an alias for checked-value. This parameter is used to implement XHTML5 form controls.
remove-value	Boolean: yes|no, 1|0, "true"|"false", "on"|"off"	no	Turns remove value mode on and off. In remove value mode, if unchecked-value is not specified, unchecking the control removes the attribute being edited. In remove value mode, if checked-value is not specified, checking the control removes the attribute being edited. If the value being edited is an element value rather than an attribute, this value is set to the empty string.

key, *value*, ..., *key*, *value* may also specify style parameters [52].

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. collapser

collapser(*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 [26].

Key	Value	Default	Description
collapsed-icon	url(), disc, circle, square, icon()	icon(collapsed-right)	Icon of the toggle button when the block or table is collapsed.
expanded-icon	url(), disc, circle, square, icon()	icon(expanded-down)	Icon of the toggle button when the block or table is expanded.
hide-when-disabled	Boolean: yes|no, 1|0, "true"|"false", "on"|"off"	no	When the toggle button is disabled (for example because the collapsible block is empty): no Render the toggle button using a gray color showing that it is currently disabled. yes Make the toggle button invisible.
show-level-menu	Boolean: yes|no, 1|0, "true"|"false", "on"|"off"	no	When specified as yes, this option adds "**Show Level 1**", "**Show Level 2**", ..., "**Show Level 9**" items to the pop-up menu displayed after right-clicking on the generated toggle button. These menu items invoke command

Key	Value	Default	Description
			"`toggleCollapsed showLevel N`" in *XMLmind XML Editor - Commands*.
tool-tip	String	No default	Tool tip text of the toggle button. A text starting with string "`<html>`" is understood to be styled using HTML (*not XHTML*). Example: "`<html>Change the  <b>linkend</b>  attribute`".

`Key`, *`value`*, ..., *`key`*, *`value`* may also specify style parameters [52], however only the color property will be honored.

Examples:

```
collapser()

collapser(tool-tip,
          "This is my tool tip!",
          color, navy)

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

collapser(collapsed-icon, url(collapsed.png),
          expanded-icon, url(expanded.png))
```

5. color-chooser

color-chooser(*`key`*, *`value`*, ..., *`key`*, *`value`*)

Inserts a button in generated content. Clicking this button displays a color chooser dialog box allowing to the user to give an hexadecimal color value (format: *`#HHHHHH`*; example: `#FF0000`) to the attribute or element being edited.

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
swatch-width	Number (implicit `px`) or CSS length	32px	Width of the color swatch displayed by the button.
swatch-height	Number (implicit `px`) or CSS length	16px	Height of the color swatch displayed by the button.

`Key`, *`value`*, ..., *`key`*, *`value`* may also specify style parameters [52].

Examples:

```
color-chooser()

color-chooser(attribute, bgcolor,
              swatch-width, 1.1em,
              swatch-height, 50)
```

6. combo-box

combo-box(*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
attribute	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.
labels	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.
values	List of strings separated by new lines ("\A ")	None (*dynamically determined by examining the data type of value to be edited*)	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 [52].

Examples:

```
combo-box(attribute, value)

combo-box(labels, "Green\A Blue\A Red",
          values, "green\A blue\A red")
```

7. 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
icon	url(), disc, circle, square, icon()	No default	Icon of the button. A button can have both a label and an icon.

Key	Value	Default	Description
text	String	No default	Label of the button. May contain newlines ("`\A `" in CSS). `Element-*()` pseudo functions are allowed here (see element-name [18]).
tool-tip	String	No default	Tool tip text of the button. A text starting with string "`<html>`" is understood to be styled using HTML (*not XHTML*). Example: "`<html>Change the  <b>linkend</b>  attribute`".
command	String	No default	Name of command triggered by the button.
parameter	String	No default	Parameter of command triggered by the button. Use number 0 to specify a null parameter.
menu	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).
menu-at-left	Boolean: yes|no, 1|0, true|false, on|off	no	Value `yes` indicates that the popup menu is to be displayed at the left of the point clicked upon. Value `no` indicates that the popup menu is to be displayed at the right of the point clicked upon.
icon-gap	Length (5px, 3em, etc)	4px	Distance between icon and label.
icon-position	right | top | bottom | left	left	Position of icon relative to the label.
select	none | element	element	By default, clicking a button selects the element having the generated content before attempting to execute the command. "`select, none`" disables this behavior.

`Key`, *`value`*, ..., *`key`*, *`value`* may also specify style parameters [52].

Simple example:

```
command-button(text, "Say hello",
               command, "alert",
               parameter, "Hello!",
```

```
               select, none,
               font-style, italic)
```

Menu syntax:

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

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

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

Menu examples:

```
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"))
```

8. command-menu

Inserts a `command-button` [58] in generated content. Unlike `command-button()`, `command-menu()` dynamically computes its menu items (typically specified using `xpath()` [90]) before displaying its popup menu.

Do not specify the `menu` parameter for this type of `command-button()`. Instead, specify `command`, `values` and optionally, `parameter` and `labels` parameters.

Key	Value	Default	Description
labels	List of strings separated by new lines ("\A ")	None (use `values` as `labels`)	Specifies the labels of the menu items.
values	List of strings separated by new lines ("\A ")	None. Required.	Specifies the *values* of the menu items. These values are passed as a parameter to the command executed by this `command-menu()`. More information below.

The popup menu items specified this way all trigger the same command. This command corresponds to the value of key `command`. However each menu item uses a specific command parameter:

- If key `parameter` has been specified, its value must contain variable `%V`. Variable `%V` is replaced by the value of a menu item.
- If key `parameter` has not been specified, the value of a menu item is used as the parameter of the command.

Unless a value has been specified for key `text` or `icon`, a `command-menu()` uses `icon(pop-down)` at its icon.

Simple example:

```
command-menu(text, "Say Hello",
             command, "alert",
             labels, "English\A Français\A Deutsch"
             values, "Hello!\A Salut!\A Tschüss!");
```

The above button displays a popup menu containing 3 items: "`English`", "`Français`", "`Deutsch`". First item invokes command `alert` with parameter "`Hello!`"; second item invokes command `alert` with parameter "`Salut!`"; third item invokes command `alert` with parameter "`Tschüss!`".

Dynamic menu items example:

```
command-menu(tool-tip, "Select a name",
  command, "putAttribute",
  parameter, "name %V",
  values,
  xpath("join(document(resolve-uri('name.list', $styleSheetURL))//item, '\A ')"))
```

where XML file `name.list` looks like this:

```
<list>
  <item>John</item>
  <item>Paul</item>
  <item>George</item>
  <item>Ringo</item>
</list>
```

See also `xpath` [90].

9. 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.styled-view.ComponentFactory` (see Chapter 7, *All stylesheet extension points*).

Example (XHTML - excerpt of `xhtml_user_styles.css`):

```
select {
    display: inline-block;
    ...
    content: component("com.xmlmind.xmleditext.xhtml.form.Select");
}
```

10. convert-button

Inserts a command-button [58] 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()
```

11. 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`".

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 [66].

Key	Value	Default	Description
attribute	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.
columns	Positive integer	Depends on `pattern`.	Width of the text field in characters.
pattern	Pattern supported by `java.text.SimpleDateFormat`	A simple pattern which depends on `data-type`.	Specifies how date is to be parsed and formatted.
language	Lower-case, two-letter codes as defined by ISO-639. Example: "es".	Language of default locale.	Participates in specifying the locale to use.
country	Upper-case, two-letter codes as defined by ISO-3166. Example: "ES".	Country of default locale.	Participates in specifying the locale to use.
variant	Vendor or browser-specific code. Example: "Traditional_WIN".	Variant of default locale.	Participates in specifying the locale to use.
data-type	date | time | dateTime | gDay | gMonthDay | gMonth | gMonthYear | gYear	date	Base data type of attribute or element value being edited. Note that default pattern for **gMonthDay** is `MM/dd` and default pattern for **gYearMonth** is `yyyy/MM`.

`Key`, *`value`*, ..., *`key`*, *`value`* may also specify style parameters [52].

Example:

```
date-field()

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

12. 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 [62], 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
attribute	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.
columns	Positive integer	Depends on `format` and on `pattern`.	Width of the text field in characters.
format	`standard` | `standard-omit-time-zone` | `pattern` | `seconds-since-epoch` | `millis-since-epoch`	`standard-omit-time-zone`	Specifies the format of the date/time value. standard Standard format of a `xs:dateTime` (W3C XML Schema standard data type), with the time zone part. Example: `2001-09-11T11:30:00Z`.

Key	Value	Default	Description
			standard-omit-time-zone Standard format of a `xs:dateTime` (W3C XML Schema standard data type), without the time zone part. Example: `2007-10-18T09:16:26`. pattern The format of the date/time value is the one specified by the `pattern` option (see below). Example: `1415 03/16/1960` (for `pattern="HHmm MM/dd/yyyy"`) seconds-since-epoch Number of seconds since January 1, 1970 GMT. A real number (`double`). Example: `1.192699294E9`. millis-since-epoch Number of milliseconds since January 1, 1970 GMT. An integer (`long`). Example: `1192699313795`.
pattern	Pattern supported by `java.text.SimpleDateFormat`	A default, short, pattern depending on the locale being used.	Specifies the format of the date. Ignored unless `format=pattern`.
language	Lower-case, two-letter codes as defined by ISO-639. Example: "es".	Language of default locale.	Participates in specifying the locale to use.
country	Upper-case, two-letter codes as defined by ISO-3166. Example: "ES".	Country of default locale.	Participates in specifying the locale to use.
variant	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 [52].

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);
}
```

13. date-picker

Similar to date-time-picker [64], 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");
}
```

14. delete-button

A convenient way of specifying `command-button [58](icon, icon(delete), command, "delete", parameter, 0)`.

15. drag-source

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

Inserts a button in generated content which executes a command (see Chapter 6, *Commands written in the Java™ programming language* in *XMLmind XML Editor - Commands*) when it is dragged. There is no need to press any modifier key (e.g. **Alt**) when a `drag-source` is dragged.

Identical to command-button [58] 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*.

XHTML example:

```
a[href][class="drag-source"]:after {
    content: " " drag-source(icon, icon(drag),
                             command, "xhtml.drag");
    vertical-align: text-top;
}
```

16. drop-site

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

Inserts a button in generated content which executes a command (see Chapter 6, *Commands written in the Java™ programming language* in *XMLmind XML Editor - Commands*) when a string is dropped on it.

Identical to command-button [58] 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 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, this object will be automatically converted to a string. 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.

XHTML example:

```
a[href][class="drop-site"]:after {
    content: " " drop-site(icon, icon(drop),
                           command, "setObject",
                           parameter, "href anyURI - '%{value}'");
    vertical-align: text-top;
}
```

17. element-label, element-name, element-local-name, element-namespace-uri

`element-label()`

Same as `element-local-name()`, but slightly formatted (e.g. capitalized) in order to make the element local name usable as a title or caption.

`element-name()`

Inserts in generated content the qualified name of the element which is the subject of the CSS rule.

`element-local-name()`

Inserts in generated content the local name of the element which is the subject of the CSS rule.

`element-namespace-uri()`

Inserts in generated content the namespace URI of the element which is the subject of the CSS rule.

DocBook 5 example: the `element-label()` is "`Abstract`", the `element-name()` is "`db:abstract`", the `element-local-name()` is "`abstract`", the `element-namespace-uri()` is "`http://docbook.org/ns/docbook`".

```
abstract:before {
    display: marker;
    content: element-label();
    font-weight: bold;
    color: #004080;
}
```

18. 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
attribute	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.
columns	Positive integer	20	Width of the text field in characters.
absolute	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).
directory	Boolean: yes|no, 1|0, "true"|"false", "on"|"off"	no	Configures the file chooser dialog box. yes Dialog box can only select directories.

Key	Value	Default	Description
			no Dialog box can only select files.
save	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.
url	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 [52].

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)
```

19. gadget

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

This pseudo-function is similar to the component [62] pseudo-function except that it creates flightweight *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.styled-view.GadgetFactory` (see Chapter 7, *All stylesheet extension points*).

Example:

```
caption:before {
    content: gadget("com.xmlmind.xmledit.cssext.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 7, *All stylesheet extension points*). Example, the following rule is used to style a “remark” inserted by a reviewer:

```
*::processing-instruction(xxe-remark) {
    display: inline;
    content: gadget("com.xmlmind.xmleditapp.cmd.diff.RemarkIndicator");
    font-size: inherit;
    background-color: transparent;
}
```

20. gauge

gauge(*key*, *value*, ..., *key*, *value*)

Inserts in generated content a simple gauge 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
min	Number	0	Lower bound of the range displayed by the gauge.
max	Number	1	Upper bound of the range displayed by the gauge.
step	Number	-	The value of the gauge is automatically adjusted to be a multiple of `step`.
low	Number	-	Specifies the range that is considered to be the “low part” of the gauge. Ignored if the gauge has a thumb.
high	Number	-	Specifies the range that is considered to be the “high part” of the gauge. Ignored if the gauge has a thumb.
optimum	Number	-	Specifies the range that is considered to be the “normal part” of the gauge. If `optimum` is less than `low`, then the “low part” is the same as the “normal part”. If `optimum` is greater than `high`, then the “high part” is the same as the “normal part”.

Key	Value	Default	Description
			Ignored if the gauge has a thumb.
thumb	Boolean: yes|no, 1|0, "true"|"false", "on"|"off"	no	If true, add a thumb which can be dragged using the mouse to change the value of the gauge.
low-color	Color	blue	Color of the “low part” of the gauge. Ignored if the gauge has a thumb.
high-color	Color	red	Color of the “high part” of the gauge. Ignored if the gauge has a thumb.
optimum-color	Color	blue	Color of the “normal part” of the gauge. Ignored if the gauge has a thumb.
width	Number (implicit `px`) or CSS length	90px	Width of the gauge.
height	Number (implicit `px`) or CSS length	15px	Height of the gauge.

`key`, *`value`*, ..., *`key`*, *`value`* may also specify style parameters [52].

Examples:

```
range {
    content: gauge(thumb, true,
                   min, -100,
                   max, 100,
                   step, 5);
}

progress {
    content: gauge(attribute, value,
                   max, 100,
                   step, 1);
}

meter {
    content: gauge(attribute, value,
                   min, -273,
                   max, 1000,
                   low, -20,
                   low-color, #80A0FF,
                   high, 55,
                   optimum, 20);
}
```

21. icon

icon(*name*|*`url`*, *`foreground`*, *`background`*, *`tool_tip`*)

Inserts a built-in image in generated content.

name

One of *name* or *url* is required.

The name of the icon must be one of the following identifiers: ○ circle, ✿ cog, ▸ collapsed-right, ↷ convert, × delete, ♦ diamond, ● disc, ▾ expanded-down, ⧉ external-link, ⧉ external-link-small, 🗋 file2, ■ filled-square, ▮ half-disc-separator, hatch1, hatch2, hatch3, hatch4, hatch5, hatch6, ◇ hollow-diamond, image, image-small, ⇢ insert-after, ⇠ insert-before, ⇳ insert, invisible, 🔑 key, ◖ left-half-disc, 🔗 left-link, ◂ left, ↰ left-target, ↵ line-break, 🔗 link, − minus, ⧉ new-window, no-image, no-image-small, ⏵ play, + plus, ▾ pop-down, ◂ pop-left, ▸ pop-right, ◢ pop-se, ▴ pop-up, ↻ replace, ➔ right-arrow, ◗ right-half-disc, 🔗 right-link, ▸ right, ↱ right-target, □ square, ↷ wrap.

url

One of *name* or *url* is required. Specifies the location of an icon.

foreground

This optional parameter allows to colorize a bitonal image. It specifies the foreground color of the icon.

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

background

This optional parameter allows to colorize a bitonal image. It specifies the background color of the icon.

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

tool_tip

Optional. Specifies a tool tip text of the icon. Default: no tool tip.

Examples:

```
icon(file2)

icon(pop-right, red)

/* Not useful -url(images/anchor.png) suffices- but works. */
icon(url(images/anchor.png))

icon(url(images/anchor.png), 0)
```

```
icon(pop-down, transparent, #80000)

icon(insert, 0, transparent, "Insert...")
```

22. insert-after-button

Inserts a command-button [58] 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()
```

23. insert-before-button

Inserts a command-button [58] 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()
```

24. insert-button

Inserts a command-button [58] 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()
```

25. insert-same-after-button

A convenient way of specifying `command-button [58](icon, icon(insert-after), command, "insertNode", parameter, "sameElementAfter")`.

26. insert-same-before-button

A convenient way of specifying `command-button [58](icon, icon(insert-before), command, "insertNode", parameter, "sameElementBefore")`.

27. 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));
}
```

28. 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() [74].

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
source	url()	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-viewports are associated to attributes or to elements.
descendant	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 `imagedata/svg:svg` or `imagedata/mml:math`. Rarely used. Most image-viewports are associated to attributes or to elements.
attribute	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 (**data-type**=**anyURI**) or directly containing image data (**data-type**=**hexBinary** or **base64Binary**).

Key	Value	Default	Description
data-type	anyURI | hexBinary | base64Binary | XML	None. If the document is conforming to a W3C XML Schema or to a RELAX NG schema, this datatype 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. **data-type**=**XML** is only allowed for elements (typically an `svg:svg` element).
gzip	Boolean: yes|no, 1|0, "true"|"false", "on"|"off"	no	Ignored unless **data-type**=**hexBinary** or **base64Binary**. If **true**, image data will be compressed with gzip before being encoded in **hexBinary** or in **base64Binary**.
viewport-width	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.
viewport-height	Length (px, mm, em, etc) or percentage	None.	Specifies the height of the viewport. A percentage is a percentage of the available space. *This is currently not supported.*
content-width	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. **scale-to-fit** means change the width of the image to fit the viewport. A *max. size* 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.
content-height	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.

Key	Value	Default	Description
			scale-to-fit means change the height of the image to fit the viewport. A *max. size* 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.
preserve-aspect-ratio	Boolean: yes|no, 1|0, "true"|"false", "on"|"off"	yes	Ignored unless **content-width** and **content-height** are both set to **scale-to-fit** or are both set to a *max. size*. If **false**, the image is scaled non-uniformly (stretched) to fit the viewport.
smooth	Boolean: yes|no, 1|0, "true"|"false", "on"|"off"	no	If **true**, quality is favored over speed when rescaling the image.
horizontal-align	left | center | right	center	Specifies how the image is to be horizontally aligned in the viewport.
vertical-align	top | middle | bottom	middle	Specifies how the image is to be vertically aligned in the viewport.
fallback-image	url(), disc, circle, square, icon()	Automatically generated. May contain an error message displayed in red.	Specifies which image to display when the normal image cannot be displayed (image format not supported, file not found, corrupted image, etc)

`key`, *`value`*, ..., *`key`*, *`value`* may also specify style parameters [52].

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:

```
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')")
    );
}
```

29. 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() [90]:

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
async	Boolean: yes|no, 1|0, "true"|"false", "on"|"off"	no	Evaluate XPath expression in a worker thread. This option is supported only when **xpath** has been specified. It may be ignored depending on the value of user preference `asyncLabelLoading` in *XMLmind XML Editor - Online Help*.
attribute	Qualified name of an attribute of the element which is the target of the rule	No default. One of **attribute** or **xpath** must be specified.	Display the value of this attribute as styled text.
dimmed	Boolean: yes|no, 1|0, "true"|"false", "on"|"off"	no	Render the generated label using a dimmed variant of foreground color.
word-wrap	Boolean: yes|no, 1|0, "true"|"false", "on"|"off"	no	If needed too, wrap the words contained in the generated label. This option is supported only when **xpath** has been specified. It is ignored when **attribute** has been specified.
xpath	Literal string specifying an XPath expression using the target of the rule as its context node	No default. One of **attribute** or **xpath** must be specified.	Display the value of this XPath expression as styled text.

`Key`, *`value`*, ..., *`key`*, *`value`* may also specify style parameters [52].

XHTML examples:

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

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

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

30. 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
attribute	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.
labels	List of strings separated by new lines ("\A ")	None (use values as labels)	Labels used for the list items. The order of labels must match the order of values.
values	List of strings separated by new lines ("\A ")	None (*dynamically determined by examining the data type of value to be edited*)	In single selection mode, clicking on list item #N sets the element or attribute value being edited to value string #N. In multiple selection mode clicking on list item #N adds/removes value string #N to/from the selected set. The value strings in the selected set are then joined using the character specified by separator (' ' by default). The resulting string is assigned to the element or attribute value being edited.
rows	Positive integer	max(10, number of values)	Maximum number of rows displayed by the list control.
selection	single | multiple	single	Specifies single or multiple selection mode.
separator	Single character string	None (values are separated by any	Character used to join selected value strings in multiple selection mode. The

Key	Value	Default	Description
		type of white space characters)	resulting string is assigned to the element or attribute value being edited.

Key, *value*, ..., *key*, *value* may also specify style parameters [52].

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, ",")
```

31. Media player

component("com.xmlmind.xmledit.cssext.media.MediaPlayerFactory", *key*, *value*, ..., *key*, *value*)

Inserts in generated content a media player similar to those found in Web browsers. This facility is currently used to render XHTML 5, DITA and DocBook 5.1 audio and video elements by embedding a media player in the styled view.

The `@media media-player` block

CSS rules referencing `component("com.xmlmind.xmledit.cssext.media.MediaPlayerFactory")` must be wrapped in a `@media media-player` block. This is needed because firstly, this corresponds to a user preference (feature `MediaPlayer` in *XMLmind XML Editor - Online Help*) and secondly, this feature requires running JavaFX®, hence a Java™ runtime having a version equal or greater than 1.8.0_40.

Key	Value	Default	Description
delegate	Qualified Java™ class name	See description.	Specified class must implement interface `com.xmlmind.xmledit.cssext.media.MediaPlayerDelegate`. By default, this class is `com.xmlmind.xmledit.cssext.media.HTMLMediaPlayerDelegate`
type	`audio`|`video`	No default.	Specify `audio` to create an audio-only media player; `video` to create a video-only media player. Do not specify type

Key	Value	Default	Description
			to create a general purpose, audio or video, media player.

XHTML 5 example:

```
@media media-player {
video {
    content: component("com.xmlmind.xmledit.cssext.media.MediaPlayerFactory",
                       type, video);
}

audio {
    content: component("com.xmlmind.xmledit.cssext.media.MediaPlayerFactory",
                       type, audio);
}
}
```

DocBook 5.1 example:

```
@media media-player {
videoobject {
    content: component("com.xmlmind.xmledit.cssext.media.MediaPlayerFactory",
                       type, video,
                       delegate,
                       "com.xmlmind.xmleditext.docbook.MediaPlayerDelegateImpl");
}

audioobject {
    content: component("com.xmlmind.xmledit.cssext.media.MediaPlayerFactory",
                       type, audio,
                       delegate,
                       "com.xmlmind.xmleditext.docbook.MediaPlayerDelegateImpl");
}
}
```

32. 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`".

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
attribute	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.
columns	Positive integer	Depends on `pattern`.	Width of the text field in characters.
country	Upper-case, two-letter codes as defined by ISO-3166. Example: "ES".	Country of default locale.	Participates in specifying the locale to use.
data-type	`byte` | `short` | `int` | `long` | `float` | `double`	`double`	Base data type of attribute or element value being edited.
language	Lower-case, two-letter codes as defined by ISO-639. Example: "es".	Language of default locale.	Participates in specifying the locale to use.
pattern	Pattern supported by `java.text.DecimalFormat`	A simple pattern which depends on **data-type**.	Specifies how number is to be parsed and formatted.
variant	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 [52].

Example:

```
number-field()

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

See also spinner [86].

33. password-field

Same as `text-field()` [89], except that the characters typed by the user are replaced by the bullet character ('•').

34. property

property(*`property_qualified_name`*)

Inserts in generated content the value of a (application-level) property of the element which is the subject of the CSS rule.

Example: topmost included elements have a `{http://www.xmlmind.com/xmleditor/namespace/property}inclusion` property. Make these properties visible in the styled view.

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

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

See also Section 14, ":property() extension pseudo class" [33].

35. 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
attribute	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.
labels	List of strings separated by new lines ("`\A `")	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.
values	List of strings separated by new lines ("`\A `")	None (*dynamically determined by examining the data type of value to be edited*)	In single selection mode, clicking on radio button #N sets the element or attribute value being edited to value string #N. In multiple selection mode clicking on check box #N adds/removes value string #N to/from the selected set.

Key	Value	Default	Description
			The value strings in the selected set are then joined using the character specified by separator (' ' by default). The resulting string is assigned to the element or attribute value being edited.
columns	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 **rows** and **columns** for the same control.
rows	Positive integer	None	Maximum number of rows used to layout the panel containing the radio buttons or check boxes. Do not specify **rows** and **columns** for the same control.
selection	single | multiple	single	Specifies single or multiple selection mode.
separator	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 [52].

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, ",")
```

36. remove-attribute-button

remove-attribute-button(**attribute**, *attribute_name*, *key*, *value*, ..., *key*, *value*)

Inserts a command-button [58] 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)
```

37. replace-button

Inserts a command-button [58] 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()
```

38. set-attribute-button

set-attribute-button(**attribute**, *`attribute_name`*, *`key`*, *`value`*, ..., *`key`*, *`value`*)

Inserts a command-button [58] 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));
```

39. spinner

spinner(*`key`*, *`value`*, ..., *`key`*, *`value`*)

Inserts in generated content a spinner 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
attribute	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.
columns	Positive integer	Automatically computed when possible; otherwise 10	Width of the text field in characters.
country	Upper-case, two-letter codes as defined by ISO-3166. Example: "ES".	Country of default locale.	Participates in specifying the locale to use.
data-type	`byte`|`short`|`int` | `long` | `float` | `double`	`double`	Base data type of attribute or element value being edited.
language	Lower-case, two-letter codes as defined by ISO-639. Example: "es".	Language of default locale.	Participates in specifying the locale to use.
max	Number	No maximum.	Maximum value for the number edited using the spinner.
min	Number	No minimum.	Minimum value for the number edited using the spinner.
pattern	Pattern supported by `java.text.DecimalFormat`	A simple pattern which depends on **data-type**.	Specifies how number is to be parsed and formatted.
step	Strictly positive number	1	The number is incremented or decremented by this amount when the user clicks the arrow buttons of the spinner.
variant	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 [52].

Example:

```
count {
    content: spinner(attribute, value,
                     data-type, byte);
}

weight {
```

```
    content: spinner(min, 0,
                     max, 1,
                     step 0.01,
                     pattern, "0.00####",
                     language, en,
                     country, "US");
}
```

See also number-field [82].

40. 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 [52].

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);
}
```

41. 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
attribute	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	Default	Description
columns	Positive integer	None (the text field expands when the document view is resized)	Width of the control in characters.
rows	Positive integer	3	Number of lines displayed by the control
wrap	none | line | word	none	Specifies how text lines are wrapped.

Key, *value*, ..., *key*, *value* may also specify style parameters [52].

Example:

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

42. 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
attribute	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.
columns	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 [52].

Example:

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

43. time-picker

Similar to date-time-picker [64], 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);
}
```

44. 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
attribute	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 [52].

Examples:

```
value-editor()

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

45. wrap-button

Same as convert-button [62] except that this button “wraps”(replaces the element by a newly created parent containing this element) the element for which the button has been generated.

46. 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() [78].

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 - Support of XPath 1.0*.

About the `document()` function when used in the context of a CSS stylesheet

Example: let's file `list.xml` contain a list of admonition types:

```
<list>
  <item>note</item>
  <item>important</item>
  <item>caution</item>
  <item>warning</item>
</list>
```

The CSS file contains the following rule:

```
warning:before {
  content: combo-box(attribute, type,
                     values,
                     xpath("join(document('list.xml')//item, '\A')"));
}
```

Because file `list.xml` is found in the same `css/` directory as the CSS stylesheet, you'll expect the above invocation of `document()` to work fine. The problem is that the above invocation of `document()` will *not* work.

Expression `document('list.xml')` means: load file `list.xml`, where URI `list.xml` is resolved against the URI of the XML node containing the invocation of `document()`. A CSS file does not contain XML, hence there is no XML node containing the invocation of `document()`.

One way to make the above example work is to replace `document('list.xml')` by `document(resolve-uri('list.xml', $styleSheetURL))`. Variable `$styleSheetURL` is automatically set by **XXE** during the evaluation of an XPath expression in the context of a CSS stylesheet and contains the URL of this CSS stylesheet.

Another way to make the above example work is to replace `document('list.xml')` by `document('my-config:css/list.xml')`, where the "`my-config:`" URI prefix is defined as follows in the *XML catalog* of your custom **XXE** configuration:

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

More information about such custom XML catalogs in XML catalogs in *XMLmind XML Editor - Configuration and Deployment*.

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 [52].

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 [52].

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 [22].

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 [52]. 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 [52] for cells *but not for rows and row*. This is different from the behavior of division [93] and paragraph [93] 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.

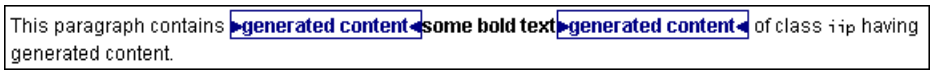
```
b.iil:before,
b.iil:after {
    display: inline;
    content: icon(right) "generated content" icon(left);
    color: navy;
}
```

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;
}
```

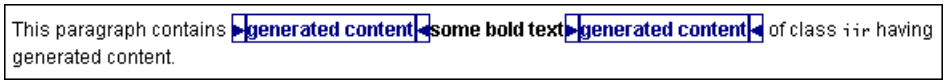


- division.

```
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;
}
```

- 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;
}
```



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

2. display: block

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

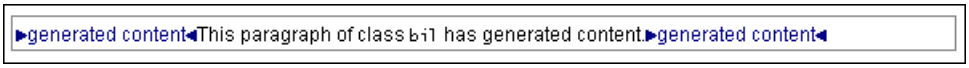
- display: inline. Supported layouts:

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

  - list.

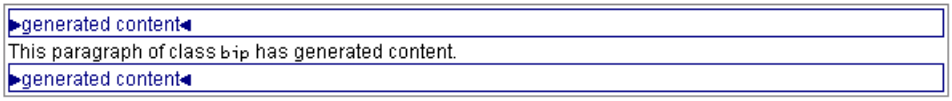
```
p.bil {
    border: 1 solid gray;
```

```
    padding: 2;
}

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



- 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.

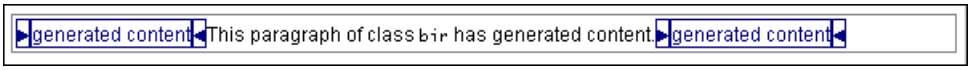
- division.

```
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;
}
```

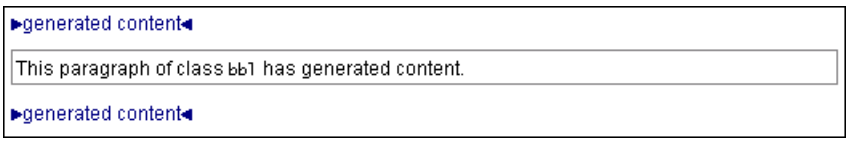


- display: block. Supported layouts:

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

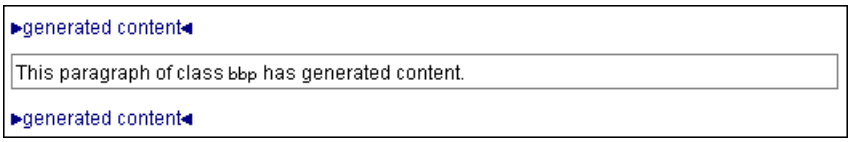
  - list.

```
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.

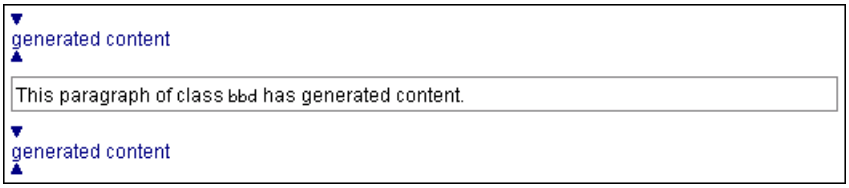
```
p.bbp {
    border: 1 solid gray;
    padding: 2;
}

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.

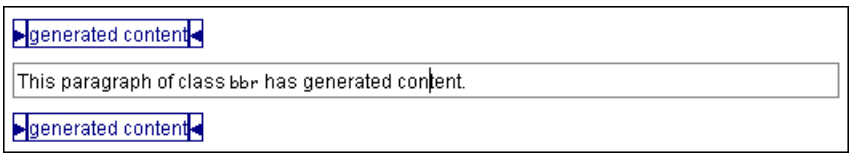
```
p.bbd {
    border: 1 solid gray;
    padding: 2;
}

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;
}
```



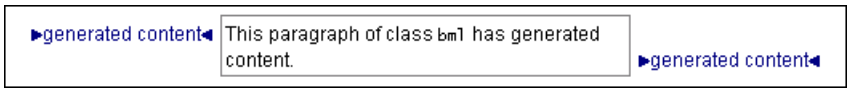
- display: marker. Supported layouts:

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

  - list.

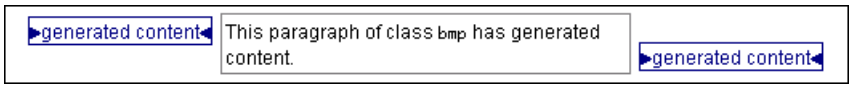
```
p.bml {
    border: 1 solid gray;
    padding: 2;
    margin-left: 20ex;
    margin-right: 20ex;
}

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

```
    color: navy;
}
```



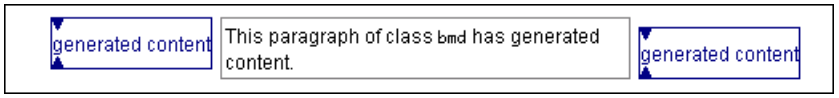
- paragraph.

```
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;
}
```



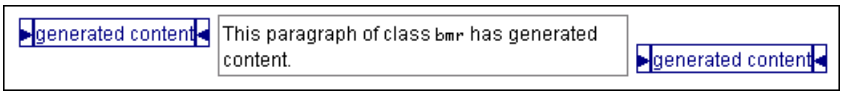
- division.

```
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 [96], 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;
}
```

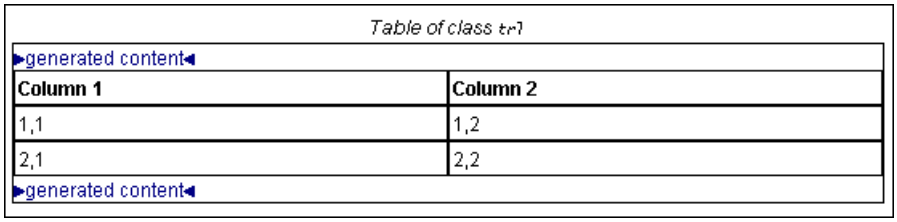
Note that if the CSS style sheet explicitly specifies a generated content before the list item, display: list-item is strictly equivalent to display: block [96] because, in such case, no content is automatically generated.

4. display: table

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

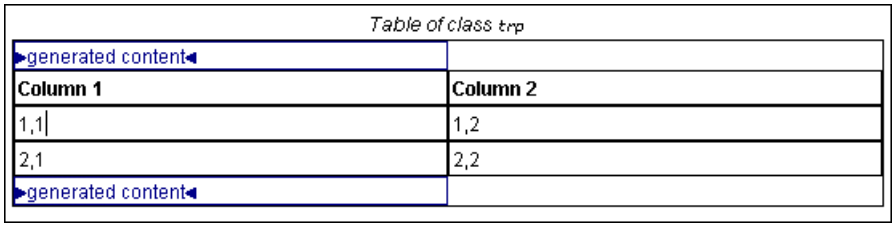
- display: block. Same behavior as display: block [96].
- display: marker. Same behavior as display: block [96].
- display: table-row-group or display: table-row. Supported layouts:
  - list.

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



  - paragraph.

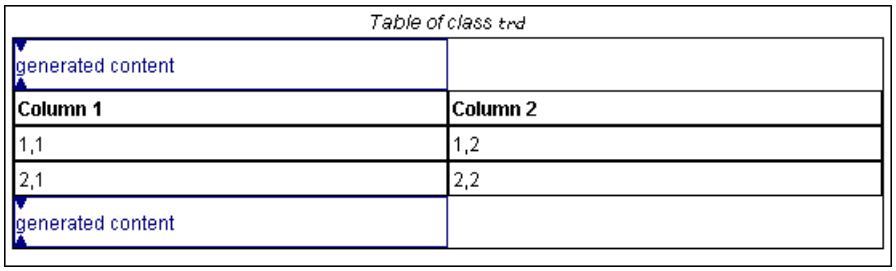
```
table.trp:before,
table.trp:after {
    display: table-row;
    content: paragraph(content(icon(right), "generated content", icon(left)),
                       border-width, 1,
                       border-style, solid);
    color: navy;
}
```



  - division

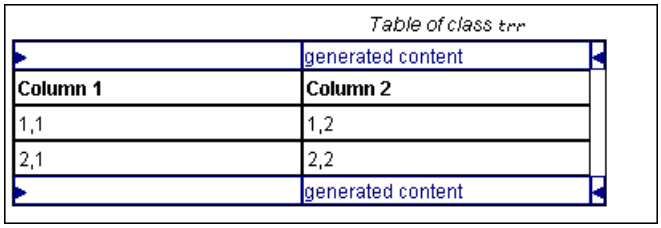
```
table.trd:before,
table.trd:after {
    display: table-row;
    content: division(content(icon(down), "generated content", icon(up)),
                      border-width, 1,
                      border-style, solid);
```

```
    color: navy;
}
```



- 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;
}
```



Note that generated row has been merged to normal content. See also rendering repeating elements as a table [22].

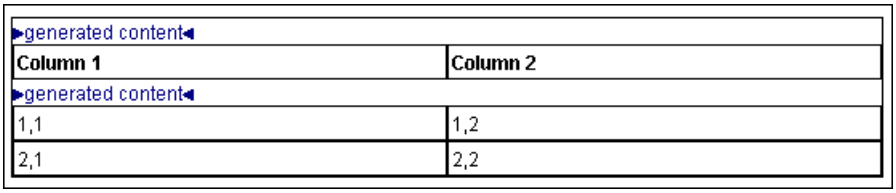
- Other display values are ignored and processed like display: block.

5. display: table-row-group

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

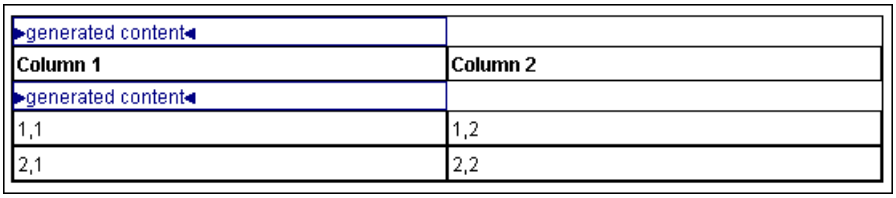
- display: table-row. Supported layouts:
  - list.

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



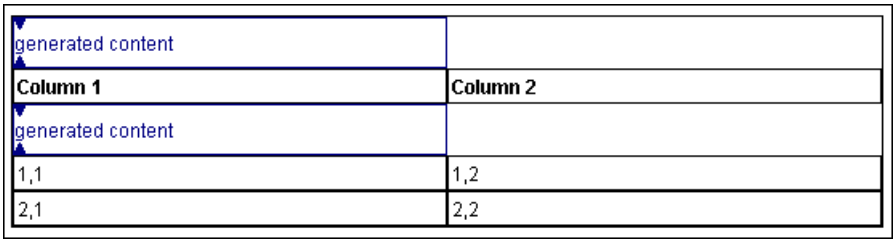
- paragraph.

```
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;
}
```



- division

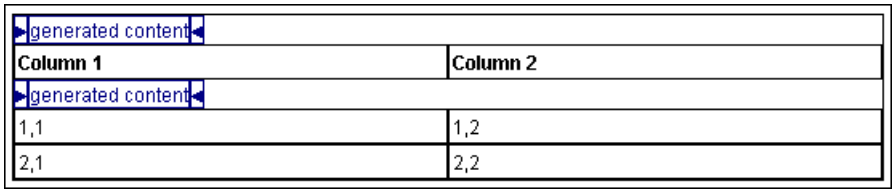
```
thead.grd:before,
thead.grd:after {
    display: table-row;
    content: division(content(icon(down), "generated content", icon(up)),
                      border-width, 1,
                      border-style, solid);
    color: navy;
}
```



- 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;
}
```



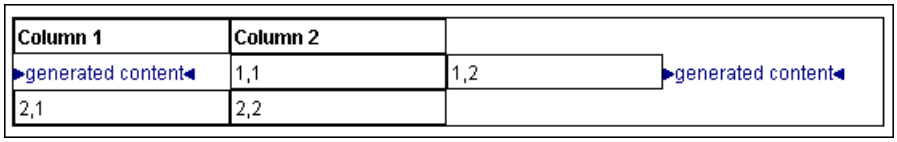
- Other display values are ignored and processed like display: table-row.

6. display: table-row

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

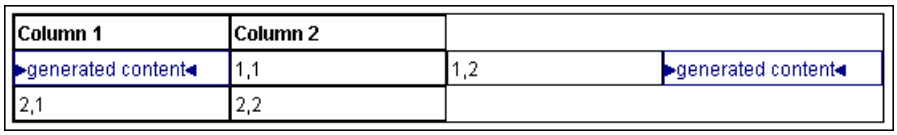
- display: table-cell. Supported layouts:
  - list.

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



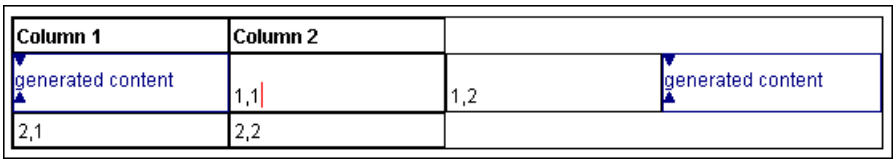
  - paragraph.

```
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;
}
```



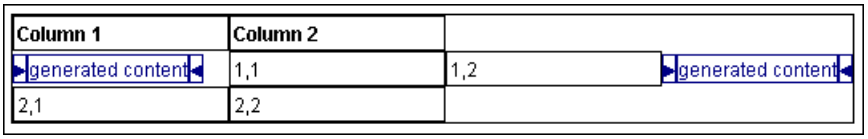
  - division

```
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;
}
```



- 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;
}
```



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

7. display: table-cell

Same behavior as display: block [96].