The

CSSToXSLFO

User Guide

Version 1.3.2
Werner Donné
Re
5th May 2006

TABLE OF CONTENTS

INTRODUCTION 1

CSSTOXSLFO is a tool which converts an XML document, combined with a CSS2 style sheet, into an XSL-FO file. It has some special provisions for XHTML, which is also an XML vocabulary. The tool implements a reasonable subset of CSS2. It also adds a few extensions for handling page-related issues properly. Note that the tool is not a general-purpose printing tool for any kind of HTML pages you can find on the Internet.

The goal of CSSTOXSLFO is to provide a rather easy interface to fine printing environments that use XSL-FO as their input. It is a compromise between the simplicity of style sheet expression and the quality of the result. XSL-FO is quite difficult. Writing style sheets that produce it are mostly written in XSLT, which is not straightforward to everyone either. CSS on the other hand is rather simple and yet it is powerful. In fact it combines element selection and formatting specification in one easy-to-learn syntax. The cost is that a lot of interesting XSL-FO features are not available.

An area where the tool can be a plus is the programmatic generation of reports within applications. The variety in style for reports is not that great. The offered feature set of CSSTOXSLFO can be sufficient. Having report programmers learn XSL-FO and XSLT is not always an option, while many know CSS and XHTML well enough to be productive with it.

Another use-case for CSSTOXSLFO is writing documents in XML. One can put work in a style sheet once and reuse that through the mark-up paradigm, in which content and formatting are separated. The formatting features should be sufficient to produce day-to-day documents in a typical business environment. Such documents don't tend to be very complicated, with respect to layout that is.

IN PRACTICE 2

2.1 SPECIFYING STYLE SHEETS

The most general way of specifying a style sheet for a document with CSSTOXSLFO is the proposal in section 2.2 of [CSS2]. It consists of a processing instruction, which precedes the document, looking like this:

```
<?xml-stylesheet type="text/css" href="style.css"?>
```

For XHTML there are a few additional options. You can use the `link` element to link a style sheet (only persistent style sheets) to the document or you can embed it with the `style` element. The `style` attribute is also available as specified in [XHTML].

2.2 RUNNING IT

There are six packages you can run from the command-line: one that produces plain XSL-FO, one that returns the output of XEP, a product from RenderX (http://www.renderx.com), another that returns the output of XSLFormatter, a product from Antenna House (http://www.antennahouse.com), yet another that returns the output of Xinc, a product from Lunasil LTD (http://www.lunasil.com) and finally, two that run FOP (http://xml.apache.org/fop/). One is for version 0.20.5 and the other for version 0.91beta. The 0.20.5 one comes with a filter that removes a few properties, which are not supported by FOP. This makes FOP complain less.

You need JDK1.3 or higher to run the packages. For 1.3 you should create a classpath with a namespace-aware XML parser and an XSLT processor. The command-lines look as follows for plain CSSTOXSLFO:

```
> java -jar css2xslfo.jar url_or_filename <options>
```

For XEP3:

```
> java -Dcom.renderx.xep.ROOT=<XEP location> -jar css2xep.jar
    url_or_filename <options>
```

For XEP4:

```
> java -Dcom.renderx.xep.CONFIG=<XEP location>/xep.xml
    -jar <XEP location>/lib/css2xep.jar url_or_filename
    <options>
```

For XSLFormatter:

```
> set dynamic library path to <XSLFormatter location>/lib
> set environment variable AH_FONT_CONFIGFILE to
    <XSLFormatter location>/etc/font-config.xml
> java -jar <XSLFormatter location>/lib/css2xsl.jar
    url_or_filename <options>
```

For Xinc:

```
> java -jar css2xinc.jar url_or_filename <options>
```

For FOP 0.20.5:

```
> java -jar css2fop.jar url_or_filename <options>
```

For FOP 0.91beta you should create a classpath with css2fopnew.jar, fop.jar and all the JAR files in the FOP lib directory. The class you have to specify is `be.re.css.CSSToFOPNew`.

Additional system properties and/or enviroment variables can be set. Please consult the product-specific documentation for this.

In order for css2xep.jar to work, you should place it in the <XEP location>/lib directory and create a link to or a copy of your XEP JAR file with the name "xep.jar". Since XEP4 the link or copy are no longer needed, because the XEP JAR file has the expected name. For css2xsl.jar to work, you should place it in <XSLFormatter location>/lib. The css2fop.jar file needs to be next to fop.jar, which should be next to the packages is uses. Therefore you should copy fop.jar from the FOP build directory to its lib directory. The css2xinc.jar should be in the XINC lib directory.

CSSTOXSLFO uses the XSLT-processor that comes with the JDK1.4, which is Xalan from Apache. For better performance you can prepend Xalan 2.6.0+ or Saxon 8.3+ to your boot classpath as follows (assuming /usr/local as the installation directory of Xalan):

```
> java -Xbootclasspath/p:/usr/local/xalan-j_2_6_0/bin/xalan.jar
    -jar css2xslfo.jar url_or_filename <options>
```

You can also use JDK1.5, which comes with a faster XSLT processor.

For XEP there is a special note. You have to specify another XSLT processor, because XEP uses Saxon 6.5.x, with which it doesn't work. You can either prepend another XSLT processor to the boot classpath or you can simply copy saxon8.jar in the XEP lib directory.

2.2.1 Common Options

The following options are common to all three variants. The document to be processed can be specified with a URL or filename. If it is omitted, stdin will be read.

-baseurl <URL>
: Change the base URL of the input document. By default it is the URL of the document itself.

-c <URL or filename>
: Specify a catalog in the format defined by SGML Open Technical Resolution TR9401:1997. Only the "PUBLIC" and "SYSTEM" keywords are supported.

-h
: Display the command-line syntax.

-p <comma-separated list of URLs or filenames>
: A list of pre-processing XSLT style sheets that are executed on the input document, in the specified order, before anything else.

-uacss <URL or filename>
: Use another User Agent style sheet than the one built-in for XHTML. Note that the latter will only be active for documents which are in the XHTML namespace (http://www.w3.org/1999/xhtml).

-v
: Turn on XML validation of the input document.

parameter=value
: Specify User Agent parameters. Equivalent CSS constructs precede these.

2.2.2 Options Specific To css2xslfo.jar

-debug
: Produces a number of intermediary files representing the different processing steps.

-fo <filename>
: The XSL-FO output file. If it is omitted stdout will be written instead.

2.2.3 Options Specific To css2xep.jar

One the following options should be specified.

-pdf <filename>
: The PDF output file.

-ps <filename>
: The PostScript output file.

2.2.4 Options Specific To css2xsl.jar

-pdf <filename>
: The PDF output file. This option is mandatory.

2.2.5 Options Specific To css2fop.jar

-fc <filename>
: An user configuration file.

-pdf <filename>
: The PDF output file.

-ps <filename>
: The PostScript output file.

-q
: Makes FOP silent.

-svg <filename>
: The SVG output file.

2.2.6 Options Specific To css2fopnew.jar

-fc <filename>
: An user configuration file.

-pdf <filename>
: The PDF output file.

-ps <filename>
: The PostScript output file.

-svg <filename>
: The SVG output file.

2.2.7 Options Specific To css2xinc.jar

One the following options should be specified.

-pdf <filename>
: The PDF output file.

2.2.8 User Agent Parameters

The User Agent parameters are common to all three packages. They have no effect if there are @page rules in the style sheet, except for the "rule-thickness" parameter. Furthermore, equivalent CSS constructs, when present in the style sheet, always precede.

column-count
: The number of columns on a page. The default is "1".

country
: The country code. The default is "GB".

font-size
: The point size of the font. The default for paper sizes "a5" and "b5" is "10pt". For all other paper sizes the default is "11pt". See also the "paper-size" parameter.

html-header-mark
: An HTML element can be passed here. Its contents will be used as the running header. By default there is no mark.

language
: The language code. The default is “en”.

odd-even-hift
: The amount by which the page contents is shifted in the inline progression direction when the paper mode is “twosided”. The default is “10mm”. See also the “paper-mode” parameter.

orientation
: The allowed values are “portrait”, which is the default, and “landscape”.

paper-margin-bottom
: The bottom margin of a page. The default is “0mm”.

paper-margin-left
: The left margin of a page. The default is “25mm”.

paper-margin-right
: The right margin of a page. The default is “25mm”.

paper-margin-top
: The top margin of a page. The default is “10mm”.

paper-mode
: The allowed values are “onesided”, which is the default, and “twosided”.

paper-size
: The allowed values are “a4”, which is the default, “a0”, “a1”, “a2”, “a3”, “a5”, “b5”, “executive”, “letter” and “legal”.

rule-thickness
: The default thickness for rules when there was no CSS specification for it. The default is “0.2pt”.

writing-mode
: The XSL-FO writing mode. The default is “lr-tb”. Other possible values are “rl-tb”, “tb-rl”, “lr”, “rl” and “tb”. See also [XSL-FO].

2.3 BUILDING CSSTOXSLFO

The tool comes with an ANT file. The default target only builds the css2xslfo.jar file. Then there are also the `xep`, `xsl`, `xinc`, `fop` and `fopnew` targets, which produce css2xep.jar, css2xsl.jar, css2xinc.jar, css2fop.jar and css2fopnew.jar respectively.

COMPLIANCE WITH CSS2 3

3.1 SPECIFICATIONS

Section	*Implemented*	*Remarks and restrictions*
4.1 Syntax	yes	Thanks to Flute.
4.2 Rules for handling parsing errors	partial	Unknown properties will end up in the XSL-FO file and cause errors in a XSL-FO processor.
4.3 Values	yes	Thanks to Flute.
4.4 CSS document representation	yes	Thanks to Flute.
5 Selectors	partial	All sections but 5.11.2 and 5.11.3. The `:first-letter` pseudo element is implemented with the restriction that letter combinations, which are considered as one letter, are not examined. As a workaround you can use the ligature Unicode characters instead. The `vertical-align` is also valid when `float` is `none`. In that case it applies to the inline material which is affected by the pseudo element.
6 Assigning property values, Cascading and Inheritance	yes	
7 Media types	yes	By design, only types `all` and `print` are supported.
8 Box model	yes	
9.1.1 The viewport	no	
9.1.2 Containing blocks	yes	

Section	*Implemented*	*Remarks and restrictions*
9.2.1 Block-level elements and block boxes	partial	Compact and run-in boxes are not supported.
9.2.2 Inline-level elements and inline boxes	partial	Compact and run-in boxes and inline tables are not supported.
9.2.3 Compact boxes	no	
9.2.4 Run-in boxes	no	
9.2.5 The 'display' property	partial	See property table.
9.3 Positioning schemes	yes	
9.4 Normal flow	yes	
9.5 Floats	yes	
9.6 Absolute positioning	yes	
9.7 Relationships between 'display', 'position', and 'float'	yes	
9.9 Layered presentation	yes	
9.10 Text direction: the 'direction' and 'unicode-bibi' properties	yes	
10 Visual formatting model details	partial	See the property table for the height property.
11 Visual effects	yes	
12.1 The :before and :after pseudo-elements	yes	
12.2 The 'content' property	yes	
12.3 Interaction of :before and :after with 'compact' and 'run-in' elements	no	
12.4 Quotation marks	yes	
12.5 Automatic counters and numbering	yes	
12.6.1 Markers: the 'marker-offset' property	yes	

Section	*Implemented*	*Remarks and restrictions*
12.6.2 Lists: the 'list-style-type', 'list-style-image', 'list-style-position', and 'list-style' properties	partial	The list-style-image property is not supported. See also the property table.
13.2.1 Page margins	yes	
13.2.2 Page size: the 'size' property	yes	
13.2.3 Crop marks: the 'marks' property	no	
13.2.4 Left, right, and first pages	yes	
13.2.5 Content outside the page box	yes	
13.3 Page breaks	partial	Named pages are only supported for block-level and table elements, which are not inside of a table and have an ancestor with the `region` property set to `body`.
13.4 Cascading in the page context	yes	
14 Colors and Backgrounds	yes	
15 Fonts	partial	@font-face and descriptors are not supported.
16 Text	yes	
17 Tables	partial	Inline tables are not supported. Anonymous table objects are only supported for missing table groups and missing table cells in a row, on the condition that there are table column elements. Audio rendering is not supported.
18 User interface	no	
19 Aural style sheets	no	

3.2 PROPERTIES

Property	*Implemented*	*Remarks and restrictions*
azimuth	no	
background	yes	
background-attachment	yes	
background-color	yes	
background-image	yes	
background-position	yes	
background-repeat	yes	
border	yes	
border-bottom	yes	
border-bottom-color	yes	
border-bottom-style	yes	
border-bottom-width	yes	
border-collapse	partial	Not for `inline-table`.
border-color	yes	
border-left	yes	
border-left-color	yes	
border-left-style	yes	
border-left-width	yes	
border-right	yes	
border-right-color	yes	
border-right-style	yes	
border-right-width	yes	
border-spacing	partial	Not for `inline-table`.
border-style	yes	

Property	*Implemented*	*Remarks and restrictions*
border-top	yes	
border-top-color	yes	
border-top-style	yes	
border-top-width	yes	
borded-width	yes	
bottom	yes	
caption-side	yes	
clear	yes	
clip	yes	
color	yes	
content	yes	
counter-increment	yes	
counter-reset	yes	
cue	no	
cue-after	no	
cue-before	no	
cursor	no	
direction	yes	
display	partial	The values `run-in`, `compact` and `inline-table` are not supported. The `marker` value is supported with the limiation that the value `auto` for the `width` property is not. Markers also don't work with floats.
elevation	no	
empty-cells	yes	
float	yes	

Property	*Implemented*	*Remarks and restrictions*
fonts	yes	
font-family	yes	
font-size	yes	
font-size-adjust	yes	
font-stretch	yes	
font-style	yes	
font-variant	yes	
font-weight	yes	
height	partial	A percentage value for the height of a block, which is in another block with an explicit height, will be treated as `auto`. This stems from the fact that in this case a block has to be split in a fo:block-container and a nested fo:block, because there are properties that don't apply to both of them. The inner original block will therefore have a parent without an explicit height specification. The latter has moved to the surrounding fo:block-container.
left	yes	
letter-spacing	yes	
line-height	yes	
list-style	partial	See individual properties.
list-style-image	no	
list-style-position	partial	A list should be uniform. Specifying different values for different list items will produce undesired results.
list-style-type	partial	Only the values `disc`, `circle`, `square`, `decimal`, `lower-`

Property	*Implemented*	*Remarks and restrictions*
		`roman`, `upper-roman`, `lower-alpha`, `lower-latin`, `upper-alpha`, `upper-latin` and `none` are supported. The additional glyphs defined in [CSS3L] are also supported. Those are `box`, `check`, `diamond` and `hyphen`. A list should be uniform. Specifying different values for different list items will produce undesired results.
margin	yes	
margin-bottom	yes	
margin-left	yes	
margin-right	yes	
margin-top	yes	
marker-offset	yes	
marks	no	
max-height	yes	
max-width	yes	
min-height	yes	
min-width	yes	
orphans	yes	
outline	no	
outline-color	no	
outline-style	no	
outline-width	no	
overflow	yes	
padding	yes	

Property	*Implemented*	*Remarks and restrictions*
padding-bottom	yes	
padding-left	yes	
padding-right	yes	
padding-top	yes	
page	partial	Only for block-level and table elements, which are not inside of a table and have an ancestor with the `region` property set to `body`.
page-break-after	yes	
page-break-before	yes	
page-break-inside	yes	
pause	no	
pause-after	no	
pause-before	no	
pitch	no	
play-during	no	
play-range	no	
position	yes	
quotes	yes	
richness	no	
right	yes	
size	yes	
speak	no	
speak-header	no	
speak-numeral	no	
speak-punctuation	no	

Property	*Implemented*	*Remarks and restrictions*
speech-rate	no	
stress	no	
table-layout	yes	
text-align	yes	
text-decoration	yes	
text-indent	yes	
text-transform	yes	
top	yes	
unicode-bibi	yes	
vertical-align	yes	
visibility	yes	
voice-family	no	
volume	no	
white-space	yes	
widows	yes	
width	yes	
word-spacing	yes	
z-index	yes	

4 EXTENSIONS

The extension features of the tool mostly pertain to page-oriented aspects. Care has been taken to not introduce new syntax. There are, however, a number of new properties. Those are normally safely ignored by browsers. In the case where there would be an impact on the layout produced by browsers, the properties can be confined to the "print" medium through @media rules.

4.1 PAGE REGIONS

This extension introduces XSL-FO-compatible page regions. Regions can be defined by placing a `region` property on an element. The allowed values are `bottom`, `left`, `right`, `top` and `body`. At least one element with the `region` property set to `body` should be present in the document.[1] Page sequences are only generated for the content of such an element. The regions other than the body region must be the first direct children of the body region. Otherwise they are ignored. In the case of XHTML, for example, this means that they should come at the beginning of the `body` element.

On top of that, either the `width` property, for left and right regions, or the `height` property, for top and bottom regions, should be defined. They will determine the dimensions of the page regions. The default value for `width` is "20mm". For `height` it is "10mm".

The extension property `precedence` is also available for the top and bottom regions. Its value can be `true` or `false`, the latter being the initial value. The property says whether the width of the top or bottom region is equal to that of the page reference area or if they give way to the left and right regions.

The regions work together with the @page rules, of which there should be at least one. It is possible to specify different regions, which correspond to the different page types in the style sheet. This can be achieved by also specifying the `page` property, which is a standard CSS2 property. Consider the following example:

```
div.bottom-left, div.bottom-right { display: none; }

@media print
{
  div.bottom-left
  {
    height: 15mm;
    page: left;
    region: bottom;
    text-align: left;
```

1 The XHTML User Agent style sheet sets this property to the `body` element.

```
    }

    div.bottom-right
    {
      height: 15mm;
      page: right;
      region: bottom;
      text-align: right;
    }

    span.page:before { content: counter(page); }
  }
```

This says that on left pages the bottom region is left-aligned, while on right pages it is right-aligned. The `span` element is used in the following region definitions:

```
<div class="bottom-left">
  <p> </p>
  <div><span class="page"/></div>
</div>

<div class="bottom-right">
  <p> </p>
  <div><span class="page"/></div>
</div>
```

The `page` property bears a kind of inheritance mechanism. For any page the regions with the most specific `page` property will be selected. A region without a `page` property is the least specific. A named page is more specific and the values `left` and `right` are yet more specific. After this comes the new pseudo page `blank`, which is for blank pages that are generated because of page positioning constraints such as left and right. The first page of a chapter, for example, is sometimes forced to be a right page. This can produce an extra blank page for the previous chapter. In fact, this maps directly to the XSL-FO blank pages. There are special values, which are even more specific, such as `first-right`, `blank-left`, `left-<page-name>`, etc. If, for example, there is no bottom region for `first-right`, but there is one for `first`, the latter will be selected if the first page happens to on the right. See section "*page*" for the precise precedence rules.

In order for the top, bottom, left and right region elements not to interfere with the normal flow it is best to set their display type to `none`.

4.2 PAGE NUMBERING

The two special counters `page` and `pages` in this tool are taken over from the CSS3 Paged Media Module (see also [CSS3P]). The `page` can be used just like any other counter, except that it is confined to the regions. The following example shows a document with a preface and a body. Each reset the page count. The preface has a lower Roman numbering style, while the body uses the decimal style. If the body page didn't reset the counter, numbering would continue from the preface, but with a change of style.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title></title>
    <style type="text/css">
@page preface
{
  counter-reset: page;
  margin: 10%;
}

@page body
{
  counter-reset: page;
  margin: 10%;
}

div.bottom-preface
{
  page: preface;
  region: bottom;
}

div.bottom-body
{
  page: body;
  region: bottom;
}

div.bottom-preface > span.page:before
{
  content: counter(page, lower-roman);
}

div.bottom-body > span.page:before
{
  content: counter(page, decimal);
}

div.preface { page: preface; }
div.body { page: body; }
    </style>
  </head>
  <body>
    <div class="bottom-preface"><span class="page"/></div>
    <div class="bottom-body"><span class="page"/></div>
    <div class="preface">
      <p>Text.</p>
    </div>
    <div class="body">
      <p>Text.</p>
    </div>
```

```
    </body>
  </html>
```

When switching between named pages you can control how the ending named page sequence should be terminated with the extension property `force-page-count`. For example, it some page sequence produces five pages, you can force the sequence to produce six pages by setting the property to `even`. An extra blank page will then be generated before starting the new page sequence. If you don't want such behaviour, you should set the property to `no-force`, since the initial value is `auto`.

4.3 PAGE REFERENCES

You sometimes want to write phrases like "The diagram on page 19 ...". The CSSTOXSLFO tool provides this functionality through the `page-ref` function, which can be used in the `content` property. Its only parameter is the name of an attribute that contains the ID of another element. The function call will be replaced with the number of the page that element is on.

In XHTML it is a bit more complicated to achieve the desired result, because there aren't many extension attributes available for it. The following fragment shows how it can be done:

```
<img id="img1" src="file:///t.png"/>
...
<span class="page-ref"><span class="img1"/></span>
```

The accompanying style sheet rule would then be:

```
span.page-ref > span:before { content: page-ref(class); }
```

4.4 LEADERS

It is possible to use XSL-FO leaders through the display type `leader`. The properties defined in section 7.21 of [XSL-FO] ("Leader and Rule Properties") can be used in a CSS style sheet, with the exception that the `leader-length` property cannot have a length range as a value. If you want to create table of contents lines or something similar, you also need the XSL-FO property `text-align-last`, described in section 7.15.10 of [XSL-FO]. The following example shows how a table of contents line could be made in XHTML.

```
<div class="toc">
  <a href="#chapter1">Title of Chapter 1</a>
  <span class="leader"/>
  <span class="page-ref"><span class="chapter1"/></span>
</div>
```

The piece of style sheet that goes with it is:

```
div.toc
{
```

```
  text-align-last: justify;
}

span.leader
{
  display: leader;
  leader-pattern: dots;
  leader-pattern-width: 5pt;
}

span.page-ref > span:before
{
  content: page-ref(class);
}
```

4.5 NAMED STRINGS

Named strings, as described in [CSS3G], are supported in CSSTOXSLFO. This consists of the `string-set` property, with which contents can be captured, and the `string()` function. The latter can occur in the value of the `content` property. The `string-set` property accepts values which are similar to those of the `content` property. There is an additional keyword `contents`, which is replaced with the string value of the element carrying the `string-set` property.

The following is a simple XHTML example of how you can create a running header that refers to the current chapter.

```
<body>
  <div class="top">
    <span class="mark"/>
  </div>
  ...
  <h1>Chapter Title</h1>
  ...
</body>
```

Here is the bit of style sheet that does it:

```
div.top
{
  region: top;
  display: none;
}

div.top > span.mark:before { content: string(mark); }

h1 { string-set: mark contents; }
```

4.6 HYPHENATION

Text can be hyphenated through the `hyphenate` property, which is inherited. The possible values are `true` and `false`. Hyphenation is turned off by default.

4.7 FOOTNOTES

It is possible to produce footnotes using the `footnote-reference` and `footnote-body` display types. The former is displayed in the flow, while the latter goes to the footnote area at the bottom of the page. When a footnote body occurs it must be either immediately preceded by a footnote reference or have a `:before` pseudo element with the display type `footnote-reference`. Otherwise it is treated as if the display were `none`. Whitespace between a footnote reference and body is gobbled. A footnote reference can also occur on its own.

The contents of both the footnote reference and body is free. Both display types exist to give you complete control over the contents and style. Usually some footnote counter is used, as shown in the example below. There is an extra counter style `footnote`, which produces symbols, such as an asterix, dagger, etc.

```
h1 { counter-reset: footnote; }

span.footnote-body
{
  display: footnote-body;
  font-size: 0.83em;
}

span.footnote-body:before
{
  content: counter(footnote);
  padding-right: 1em;
}

span.footnote-reference
{
  display: footnote-reference;
}

span.footnote-reference:before
{
  counter-increment: footnote;
  content: counter(footnote);
  font-size: 0.83em;
  vertical-align: super;
}
```

In the document a footnote would then look like this:

```
<p>Paragraph text.<span class="footnote-reference"/><span
class="footnote-body">Footnote text.</span></p>
```

You might find it cumbersome to have to place a footnote reference in front of every footnote body. It can be avoided, at the expense of formatting control however. You can define a `:before` pseudo element for the footnote body and give it the display type `footnote-reference`. Whatever contents it generates will then be used for the reference in the flow, as well as in the footnote body at the bottom of the page. As

a consequence, the style is constrained by the fact that it must be decent for both contexts. The style sheet becomes a bit simpler:

```
h1 { counter-reset: footnote; }

span.footnote-body
{
  display: footnote-body;
  font-size: 0.83em;
}

span.footnote-body:before
{
  counter-increment: footnote;
  content: counter(footnote);
  display: footnote-reference;
  font-size: 0.83em;
  vertical-align: super;
}
```

If you want full control over the formatting in both contexts and at the same time want to omit the footnote reference elements in the document, the solution is to preprocess the document. The transformation is rather trivial.

4.8 ORIENTATION

You can rotate text with the `orientation` property. This works only for block-level elements. The possible values are 0, 90, 180, 270, -90, -180, -270. They represent the degrees in the counter-clockwise direction. The initial value is 0.

4.9 LIST STYLE TYPES

The glyphs for the `list-style-type` property, as defined in [CSS3L], are implemented.

4.10 MULTICOLUMN

With the properties `column-count`, which must be strictly positive, and `column-gap`, which is a length, a multi-column layout can be specified for a page. Both properties are allowed in an @page rule. As a consequence, if you want to switch between column modes, you have to switch pages as well.

With the `column-span` property a blocks and tables, that are not themselves inside of another table, can be made to span all the columns of a multi-column page. The allowed values for the property are `all` and `none`.

4.11 CHANGE BARS

The change bar properties introduced in [XSL-FO11] are available for `:before` and `:after` pseudo elements. For the latter, only the `change-bar-class` property is relevant. The following is a simple example:

```
p.changed:before
{
  change-bar-class: changed;
  change-bar-style: solid; /* initial value is none */
  change-bar-width: 0.2pt;
}

p.changed:after { change-bar-class: changed; }
```

Note that this feature only works for XEP4 at the moment.

4.12 LINKS

The `link` property can have the name of an attribute as its value. The value of that attribute will be used for the generated link, as the target URL or the internal target ID, if it is an IDREF attribute, which distinguishes it from a relative URL. Likewise, the value of the `anchor` property can be the name of an attribute, the value of which will become the target ID. This way an internal link destination can be created. For example:

```
a[href] { link: href; }
a[name] { anchor: name; }
```

4.13 GRAPHICS

An external graphic can be included in a document through the display type `graphic`. The XSL-FO properties `content-height`, `content-width`, `content-type`, `scaling` and `scaling-method` are supported. Their definition is in [XSL-FO]. The property `src` is interpreted differently. Its value should be the name of an attribute that has a URI for a value. For the XHTML element `img`, for example, the User Agent style sheet contains the following:

```
img
{
  content-height: scale-to-fit;
  content-width: scale-to-fit;
  display: graphic;
  scaling: uniform;
  src: src;
}
```

4.14 COLUMN AND ROW SPANNING

In XHTML one can specify column and row spanning with the `colspan` and `rowspan` attributes on the `td` and `th` elements. It is, however, also possible to apply CSS to other XML vocabularies. Hence, there should be an equivalent feature in CSS to express this. The extension properties `colspan` and `rowspan` serve that purpose. They can be used for elements with the display type `table-cell`.

4.15 PROPORTIONAL COLUMN WIDTHS

Again in XHTML it is possible to say that a column should occupy a relative portion of the total table width. It is done by setting the `width` attribute to a number, followed by an asterix. If we have, for example, three columns with the widths "1*", "2*" and "3*", they occupy 1, 2 and 3 sixth of the table width respectively. This is not part of the HTML specification, but it is a widely supported feature.

In order to provide it for other XML vocabularies then XHTML, the unit `pcw`, which stands for "proportional column width", is available for the `width` property of an element with the display type `table-column`.

4.16 REPEATING TABLE HEADERS AND FOOTERS

By default table headers and footers are repeated when a table spans several pages. You can suppress this by setting the `table-omit-header-at-break` and `table-omit-footer-at-break` properties to `true` respectively.

4.17 CSS3 NAMESPACES

Namespaces for selectors, as defined in [CSS3S], are implemented. This means you can use namespace prefixes in element selectors and attribute conditions. The prefixes are separated from the local name with a pipe sign ("|").

The namespaces are declared with the `@namespace` rule, which should always come right after the `@import` rules if there are any. In the following example the XHTML namespace has been declared as the default namespace. Next to that, the DeltaXML namespace is declared with the prefix "dx". You also see the use of the "attr" function with an attribute that has a prefix.

```
@namespace url(http://www.w3.org/1999/xhtml);
@namespace dx
  url(http://www.deltaxml.com/ns/well-formed-delta-v1);

*[dx|delta=add], dx|new
{
  text-decoration: underline;
}

*[dx|delta=delete], dx|old
{
```

```
    text-decoration: line-through;
  }

  p[dx|delta]:before
  {
    content: attr(dx|delta);
    display: marker;
    marker-offset: 0.5em;
    text-align: right;
  }
```

4.18 WRAPPERS

When processing XML in general you might encounter elements which represent pure structure, i.e. they are not directly related to layout. For such elements there shouldn't be any formatting objects in the output. Normally you would have to pre-process the document in order to get rid of them in the proper way.

The display type `wrapper` is introduced to cope with common cases. When an element has this display type, it will not contribute any formatting objects. However, its inherited properties will be passed on to its child elements, according to the property inheritance rules.

With respect to XML processing, a wrapper seems to be "transparant". Note however that, while a wrapper can occur anywhere, it influences CSS selector matching. For instance, it will interfere with "direct sibling" and "direct child" selectors.

4.19 FOREIGN ELEMENTS

With the display type `foreign` it is possible to transfer part of a document unmodified to an `fo:instream-foreign-object` element. This may be useful for elements that are in another namespace than that of the document itself and which are supported by the XSL-FO processor. Typical examples are SVG and MathML.

4.20 PROPERTY SPECIFICATIONS

4.20.1 anchor

Value:	<identifier> | attr(X)
Initial:	none
Applies to:	block-level and inline-level elements
Inherited:	no
Percentages:	N/A
Media:	print

<identifier>
: The qualified name of an attribute, the value of which is the target ID. This type of value is *deprecated*, because it doesn't support namespace prefixes.

attr(X)
: This returns the value of the attribute of the subject with the qualified name X. The CSS3 namespace prefixes are supported. The value is the target ID.

4.20.2 change-bar-class

Value: <name>
Initial: none, value required
Applies to: before and after pseudo elements
Inherited: no
Percentages: N/A
Media: print

<name>
: An NCName, as defined in [NAMES], to allow pairing of before and after elements, which don't have to belong to the same element. This way a change bar context is created.

4.20.3 change-bar-color

Value: <color>
Initial: the value of the `color` property
Applies to: before pseudo elements
Inherited: no
Percentages: N/A
Media: print

<color>
: Specifies the color of the change bar.

4.20.4 change-bar-offset

Value: <length>
Initial: 6pt
Applies to: before pseudo elements
Inherited: no
Percentages: N/A
Media: print

<length>
: Gives the distance from the edge of the column area containing the text that is marked as changed to the center of the generated change bar. A positive distance is directed away from the column region and into the margin regardless of the `change-bar-placement` property. Relative lengths (i.e., percentage values and lengths with units of "em") are not permitted for the value of this property.

4.20.5 change-bar-placement

Value: left | right | inside | outside | alternate
Initial: start
Applies to: before pseudo elements
Inherited: no
Percentages: N/A
Media: print

alternate
: When there are exactly two columns, the change bar will be offset from the left edge of all column one areas and the right edge of all column two areas; when there are any other number of columns, this value is equivalent to "outside".

inside
: If the page binding edge is on the left-edge, the change bar will be offset from the left edge of all column areas. If the binding is the right-edge, the change bar will be offset from the right edge of all column areas. If the page binding edge is on neither the left-edge nor right-edge, the change bar will be offset from the left edge of all column areas.

left
: The change bar will be offset from the left edge of all column areas.

outside
: If the page binding edge is on the left-edge, the change bar will be offset from the right edge of all column areas. If the binding is the right-edge, the change bar will be offset from the left edge of all column areas. If the page binding edge is on neither the left-edge nor right-edge, the change bar will be offset from the right edge of all column areas.

right
: The change bar will be offset from the right edge of all column areas.

4.20.6 change-bar-style

Value: <border-style>
Initial: none
Applies to: before pseudo elements
Inherited: no
Percentages: N/A
Media: print

4.20.7 change-bar-width

Value: <border-width>
Initial: medium
Applies to: before pseudo elements
Inherited: no
Percentages: N/A

Media: print

<border-width>
: Relative lengths (i.e., percentage values and lengths with units of "em") are not permitted for the value of this property.

4.20.8 colspan

Value: <integer>
Initial: 1
Applies to: table cells
Inherited: no
Percentages: N/A
Media: print

<integer>
: Expresses the number of columns the table cell will span. The value must be larger than or equal to 1.

4.20.9 column-count

Value: <integer>
Initial: 1
Applies to: the page context
Inherited: no
Percentages: N/A
Media: print

<integer>
: The value must be larger than or equal to 1.

4.20.10 column-gap

Value: <length> | <percentage>
Initial: 12.0pt
Applies to: the page context
Inherited: no
Percentages: refer to the width of the body region
Media: print

<length>
: This is an unsigned length, If a negative value has been specified a value of opt will be used.

<percentage>
: The value is a percentage of the width of the body region.

4.20.11 column-span

Value: none | all
Initial: none
Applies to: block elements which are not in table elements
Inherited: no
Percentages: N/A
Media: print

all
: This element spans all columns of a multi-column region.

none
: This element does not span multiple columns of a multi-column region.

4.20.12 content-height

Value: auto | scale-to-fit | <length> | <percentage>
Initial: auto
Applies to: graphic elements
Inherited: no
Percentages: intrinsic height
Media: print

auto
: The content-height should be the intrinsic content-height.

scale-to-fit
: A size of the content-height equal to the height of the viewport. This implies a certain scaling factor to be applied onto the content.

<length>
: An absolute size for the content-height. This implies a certain scaling factor to be applied onto the content.

<percentage>
: A percentage representing a scaling factor applied to the intrinsic height.

4.20.13 content-type

Value: auto | <string>
Initial: auto
Applies to: graphic elements
Inherited: no
Percentages: intrinsic height
Media: print

auto
: No identification of the content-type. The User Agent may determine it by "sniffing" or by other means.

<string>
: A specification of the content-type in terms of a mime-type, which has the form "content-type:" followed by a mime content-type, e.g., content-type="content-type:image/svg+xml".

4.20.14 content-width

Value: auto | scale-to-fit | <length> | <percentage>
Initial: auto
Applies to: graphic elements
Inherited: no
Percentages: intrinsic width
Media: print

auto
: The content-width should be the intrinsic content-width.

scale-to-fit
: A size of the content-width equal to the width of the viewport. This implies a certain scaling factor to be applied onto the content.

<length>
: An absolute size for the content-width. This implies a certain scaling factor to be applied onto the content.

<percentage>
: A percentage representing a scaling factor applied to the intrinsic width.

4.20.15 display

This section specifies additional values for the property.

Value: footnote-body | footnote-reference | foreign | graphic | leader | wrapper
Initial: inline
Applies to: all elements
Inherited: no
Percentages: N/A
Media: print

footnote-body
: The contents of the element goes to the footnote area. The element must be either immediately preceded by an element of type `footnote-reference` or have a `:before` pseudo element of that type. Otherwise it is treated as if its display type were `none`. Whitespace between a footnote reference and body is removed. In case a pseudo element is used, the contents it generates is displayed in the flow, as well as in the footnote body.

footnote-reference
: This is an inline variant. Its contents is displayed in the flow. It can occur without a following `footnote-body` element.

foreign
: If an element has this display type, it is placed unmodified in an `fo:in-stream-foreign-object` element.

graphic
: This display type is used to include external graphics.

leader
: This display type is used to produce XSL-FO leaders.

wrapper
: An element with this display type doesn't contribute any formatting objects. Its inherited properties are nevertheless inherited by its subtree.

4.20.16 force-page-count

Value:	auto | even | odd | end-on-even | end-on-odd | no-force
Initial:	auto
Applies to:	the page context
Inherited:	no
Percentages:	N/A
Media:	print

The property is used to impose a constraint on the number of pages in a page sequence. In the event that this constraint is not satisfied, an additional page will be added to the end of the sequence. This page becomes the "last" page of that sequence.

auto
: Force the last page in this page sequence to be an odd page if the initial page number of the next page sequence is even. Force it to be an even page if the initial page number of the next page sequence is odd. If there is no next page sequence or if the value of its initial page number is "auto" do not force any page.

even
: Force an even number of pages in this page sequence.

odd
: Force an odd number of pages in this page sequence.

end-on-even
: Force the last page in this page sequence to be an even page.

end-on-odd
: Force the last page in this page sequence to be an odd page.

no-force
: Do not force either an even or an odd number of pages in this page sequence.

4.20.17 hyphenate

Value:	false | true
Initial:	false
Applies to:	block-level and inline-level elements
Inherited:	yes
Percentages:	N/A

Media: print

false
: Hyphenation is not active for the text in this element.

true
: Hyphenation is active for the text in this element.

4.20.18 leader-alignment

Value: none | reference-area | page
Initial: none
Applies to: leader elements
Inherited: yes
Percentages: N/A
Media: print

Specifies whether leader elements having identical content and property values shall have their patterns aligned with each other, with respect to their common reference-area or page. For leader elements where the `leader-pattern` property is specified as `dots` or as `use-content`, this property will be honored. If the leader elements is aligned, the left-edge of each cycle of the repeated pattern will be placed on the left-edge of the next cycle in the appropriate pattern-alignment grid.

none
: Leader-pattern has no special alignment.

page
: Leader-pattern is aligned as if it began on the current page's left-edge.

reference-area
: Leader-pattern is aligned as if it began on the current reference-area's content-rectangle left-edge.

4.20.19 leader-length

Value: <length> | <percentage>
Initial: 12.0pt
Applies to: leader elements
Inherited: yes
Percentages: refer to the width of the content-rectangle of the parent area.
Media: print

<length>
: Sets the length of a leader element.

<percentage>
: Sets the length of a leader element to a percentage of the width of the content-rectangle of the parent area.

4.20.20 leader-pattern

Value: space | rule | dots | use-content
Initial: space
Applies to: leader elements
Inherited: yes
Percentages: N/A
Media: print

dots
: Leader is to be filled with a repeating sequence of dots. The choice of dot character is dependent on the user agent.

rule
: Leader is to be filled with a rule. If this choice is selected, the `rule-thickness` and `rule-style` properties are used to set the leader's style.

space
: Leader is to be filled with blank space.

use-content
: Leader is to be filled with a repeating pattern as specified by the children of the leader element.

4.20.21 leader-pattern-width

Value: use-font-metrics | <length> | <percentage>
Initial: use-font-metrics
Applies to: leader elements
Inherited: yes
Percentages: refer to the width of the content-rectangle of the parent area.
Media: print

use-font-metrics
: Use the width of the leader-pattern as determined from its font metrics.

<length>
: Sets the length for leader-pattern-repeating. The leader will have an inline-space inserted after each pattern cycle to account for any difference between the width of the pattern as determined by the font metrics and the width specified in this property. If the length specified is less than the value that would be determined via the `use-font-metrics` choice, the value of this property is computed as if `use-font-metrics` choice had been specified.

<percentage>
: Sets the length for leader-pattern-repeating to a percentage of the width of the content-rectangle of the parent area.

For leader elements where the `leader-pattern` property is specified as `dots` or as `use-content`, this property will be honored.

4.20.22 link

Value: <identifier> | attr(X)
Initial: none
Applies to: block-level and inline-level elements
Inherited: no
Percentages: N/A
Media: print

<identifier>
: The qualified name of an attribute, the value of which is either a target ID or a URI. It is considered as an ID if the attribute is of type IDREF. This way a distinction is made with a relative URL. Note that the attribute type information should be available. This requires a document type definition. This type of value is *deprecated*, because it doesn't support namespace prefixes.

attr(X)
: This returns the value of the attribute of the subject with the qualified name X. The CSS3 namespace prefixes are supported. The value is considered as an ID if the attribute is of type IDREF. This way a distinction is made with a relative URL. Note that the attribute type information should be available. This requires a document type definition.

4.20.23 list-style-type

This section specifies additional glyph values for the property.

Value: box | check | diamond | hyphen
Initial: disc
Applies to: elements with “display: list-item”
Inherited: yes
Percentages: N/A
Media: print

box
: A hollow square.

check
: A check mark.

diamond
: A filled diamond.

hyphen
: A hyphen bullet.

4.20.24 orientation

Value: 0 | 90 | 180 | 270 | -90 | -180 | -270
Initial: 0
Applies to: block elements

Inherited: yes
Percentages: N/A
Media: print

0
: The material in this element is not rotated.

90
: The material in this element is rotated 90 degrees counter-clockwise with respect to the containing block element.

180
: The material in this element is rotated 180 degrees counter-clockwise with respect to the containing block element.

270
: The material in this element is rotated 270 degrees counter-clockwise with respect to the containing block element.

-90
: The material in this element is rotated 90 degrees clockwise with respect to the containing block element.

-180
: The material in this element is rotated 180 degrees clockwise with respect to the containing block element.

-270
: The material in this element is rotated 270 degrees clockwise with respect to the containing block element.

4.20.25 page

This section specifies the property in the context of static regions. It defines the pages to which the static region applies. If more than one static region of the same kind (left, right, top or bottom) applies to a page, the most specific is selected, i.e. the one for which the most conditions are fulfilled. Each property value expresses a number of conditions.

Value: auto | first | left | right | blank | first-left | first-right | blank-left | blank-right | first-<identifier> | left-<identifier> | right-<identifier> | blank-<identifier> | first-left-<identifier> | first-right-<identifier> | blank-left-<identifier> | blank-right-<identifier> | <identifier>
Initial: auto
Applies to: static regions
Inherited: no
Percentages: N/A
Media: print

auto
: Applies to any page.

blank
: Applies if the page is a blank page. Blank pages can be generated, for example, when page breaks are forced to left or right pages.

blank-left
: Applies if the page is a blank and a left page.

blank-right
: Applies if the page is a blank and a right page.

blank-<identifier>
: Applies if the page is a blank and a named page, with the name set to the specified identifier.

blank-left-<identifier>
: Applies if the page is a blank, left and named page, with the name set to the specified identifier.

blank-right-<identifier>
: Applies if the page is a blank, right and named page, with the name set to the specified identifier.

first
: Applies if the page is a first page.

first-left
: Applies if the page is a first and a left page.

first-right
: Applies if the page is a first and a right page.

first-<identifier>
: Applies if the page is a first and a named page, with the name set to the specified identifier. When the document switches to a named page sequence, using the `page` property in the regular way, the first page of that sequence is a first page.

first-left-<identifier>
: Applies if the page is a first, left and named page, with the name set to the specified identifier.

first-right-<identifier>
: Applies if the page is a first, right and named page, with the name set to the specified identifier.

left
: Applies if the page is a left page.

left-<identifier>
: Applies if the page is a left and a named page, with the name set to the specified identifier.

right
: Applies if the page is a right page.

right-<identifier>
: Applies if the page is a right and a named page, with the name set to the specified identifier.

<identifier>
: Applies if the page is a named page, with the name set to the specified identifier.

4.20.26 precedence

Value: false | true
Initial: false
Applies to: static top and bottom regions
Inherited: no
Percentages: N/A
Media: print

false
: The width of the region is reduced by the incursions of the left and right regions.

true
: The height of the left and right regions is reduced by the incursions of this region.

4.20.27 region

Value: body | left | right | top | bottom | none
Initial: none
Applies to: all elements, but see prose
Inherited: no
Percentages: N/A
Media: print

body
: There should be one element with this value for the property. For the contents of this element the page sequences will be generated.

bottom
: This element becomes the bottom static region. The pages for which this is the case can be limited through the `page` property.

left
: This element becomes the left static region.

none
: The element is not a region.

right
: This element becomes the right static region.

top
: This element becomes the top static region.

The static region elements should be the first child elements of the body region. In other words, they should precede all elements which are not static regions, otherwise their `region` property is ignored. The property is also ignored if there are no @page rules. In that case the default page set-up is generated.

4.20.28 rowspan

Value: <integer>
Initial: 1

Applies to: table cells
Inherited: no
Percentages: N/A
Media: print

<integer>
: Expresses the number of rows the table cell will span. The value must be larger than or equal to 1.

4.20.29 rule-style

This property applies only if the `leader-pattern` property is specified as `rule`.

Value: none | dotted | dashed | solid | double | groove | ridge
Initial: solid
Applies to: leader elements
Inherited: yes
Percentages: N/A
Media: print

dashed
: The rule is a series of short line segments.

dotted
: The rule is a series of dots.

double
: The rule is two solid lines. The sum of the two lines and the space between them equals the value of the `rule-thickness` property.

groove
: The rule looks as though it were carved into the canvas. (Top/left half of the rule's thickness is the color specified; the other half is white.)

none
: No rule, forces `rule-thickness` to 0.

ridge
: The opposite of "groove", the rule looks as though it were coming out of the canvas. (Bottom/right half of the rule's thickness is the color specified; the other half is white.)

solid
: The rule is a single line segment.

4.20.30 rule-sthickness

This property applies only if the `leader-pattern` property is specified as `rule`.

Value: <length>
Initial: 1.0pt
Applies to: leader elements
Inherited: yes

Percentages: N/A
Media: print

<length>
: The rule-thickness is always perpendicular to its length-axis. The rule is thickened equally above and below the line's alignment position.

4.20.31 scaling

Value: uniform | non-uniform
Initial: uniform
Applies to: graphic elements
Inherited: no
Percentages: intrinsic width
Media: print

non-uniform
: Scaling need not preserve the intrinsic aspect ratio.

uniform
: Scaling should preserve the intrinsic aspect ratio.

4.20.32 scaling-method

Value: auto | integer-pixels | resample-any-method
Initial: auto
Applies to: graphic elements
Inherited: no
Percentages: intrinsic width
Media: print

auto
: The User Agent is free to choose either resampling, integer scaling, or any other scaling method.

integer-pixels
: The User Agent should scale the image such that each pixel in the original image is scaled to the nearest integer number of device-pixels that yields an image less-then-or-equal-to the image size derived from the content-height, content-width, and scaling properties.

resample-any-method
: The User Agent should resample the supplied image to provide an image that fills the size derived from the `content-height`, `content-width`, and `scaling` properties. The user agent may use any sampling method.

4.20.33 src

Value: <identifier> | attr(X)
Initial: none, value required

Applies to: graphic elements
Inherited: no
Percentages: N/A
Media: print

<identifier>
: The qualified name of an attribute, the value of which is a URI. This type of value is *deprecated*, because it doesn't support namespace prefixes.

attr(X)
: This returns the value of the attribute of the subject with the qualified name X. The CSS3 namespace prefixes are supported. The value is a URI.

4.20.34 string-set

Value: none | <identifier> contents | <identifier> <content>
Initial: none
Applies to: all elements
Inherited: no
Percentages: N/A
Media: print

none
: No named string is set.

<identifier> contents
: The string named by the identifier is set to the textual contents of the element.

<identifier> <content>
: The string named by the identifier is set to the result of the evaluation of the expression in <content>. The syntax for the expression is the same as that for the `content` property.

4.20.35 text-align-last

Value: <identifier>
Initial: relative
Applies to: block elements
Inherited: no
Percentages: N/A
Media: print

center
: Specifies that the contents is to be centered horizontally.

inside
: If the page binding edge is on the left-edge, the alignment will be left. If the binding is on the right-edge, the alignment will be right. If neither, use left alignment.

justify
: Specifies that the contents is to be expanded to fill the available width.

left
: Specifies that the contents is to be aligned on the left-edge.

outside
: If the page binding edge is on the left-edge, the alignment will be right. If the binding is on the right-edge, the alignment will be left. If neither, use right alignment.

relative
: If `text-align` is `justify`, then the alignment of the last line, and of any line ending in U+000A, will be left. If `text-align` is not `justify`, `text-align-last` will use the value of `text-align`.

right
: Specifies that the contents is to be aligned on the right-edge.

4.20.36 table-omit-footer-at-break

Value: false | true
Initial: false
Applies to: tables
Inherited: no
Percentages: N/A
Media: print

false
: This property specifies that the footer should not be ommitted.

true
: This property specifies that the footer should be ommitted.

4.20.37 table-omit-header-at-break

Value: false | true
Initial: false
Applies to: tables
Inherited: no
Percentages: N/A
Media: print

false
: This property specifies that the header should not be ommitted.

true
: This property specifies that the header should be ommitted.

4.21 MISCELLANEOUS SPECIFICATIONS

4.21.1 The :blank Pseudo-class

The :blank pseudo-class is available to specify properties in the page context for blank pages. Those can be generated, for example, when pages are forced to start at the left or right.

4.21.2 The page And pages Counters

The page counter represents the current page number, while the pages counter represents the total number of pages in the document. Both can be used in static regions only. The page counter may be reset in the page context.

4.21.3 The page-ref Function

The page-ref function can be used in the `content` property. Its only parameter is either the qualified name of an attribute that contains the ID of another element, or the attr(X) function, where X is the qualified name of such an attribute. The former is deprecated, because it doesn't support CSS3 namespace prefixes, while the latter does. The function call will be replaced with the number of the page the target element is on.

4.21.4 The string Function

The string function produces the string that was saved with a `string-set` property. Its argument is the name used in a `string-set` property. If a named string is set more than once on a page, the first occurrence will be returned by the string function.

4.21.5 The footnote Counter Style

This counter style produces symbols in the following order: *, †, ‡, §, ||, ¶, #, **, ††, ‡‡, §§. If the counter value if larger than the number of symbols in the preceding list, the * symbol is generated.

4.21.6 The pcw Unit

This unit is available for the `width` property of an element with display type `table-column`. It expresses the proportional width for a table column. The value should be divided by the sum of all the present proportional widths, which itself is equal to the width of the table minus all fixed column widths.

4.21.7 The @namespace Rule

With the @namespace rule a namespace can be declared, with or without a prefix. In the latter case it is the default namespace. The scope of a declared namespace is limited

to the style sheet entity in which it is declared. The @namespace rule should come right after the @import rules if there are any and before all other rules. An @namespace rule has an optional prefix argument, which is an identifier, followed by a mandatory URI specification. Consult [CSS3S] to learn how namespaces work with selectors.

EMBEDDING IN AN APPLICATION 5

5.1 API SPECIFICATION

5.1.1 be.re.css.CSSToXSLFOFilter

This class extends org.xml.sax.helpers.XMLFilterImpl. The source of the SAX events that are sent through the filter must be namespace aware. If it is an XML parser this option must be turned on.

Constructors

public CSSToXSLFOFilter
(
java.net.URL baseUrl,
java.net.URL userAgentStyleSheet,
java.util.Map userAgentParameters,
XMLReader parent,
boolean debug
) throws CSSToXSLFOException

baseUrl
: The `baseUrl` is used to resolve relative URLs. This includes references to style sheets as well as other resources. Because of the latter the resulting XSL-FO document is not relocatable. The relative URLs in the input document are transformed into absolute ones using the `baseUrl`. This copes with the case where the input document is an anonymous stream that refers to style sheets in a relative way. The set of style sheets is then relocatable through the `baseUrl`.

  If `baseUrl` is `null` no resolution of relative URLs will be done.

userAgentStyleSheet
: The default style sheet for the filter. It is used in the way described in section 6.4 of [CSS2]. If the parameter is `null`, a default style sheet for XHTML is used.

userAgentParameters
: These are defined in section "*User Agent Parameters*".

parent
: The parent filter. It must not be `null`.

debug
: If `debug` is `true` a number of debug files are dumped. They show the results of the internal processing steps.

public CSSToXSLFOFilter
(
java.net.URL baseUrl,
java.net.URL userAgentStyleSheet,
java.util.Map userAgentParameters,
XMLReader parent
) throws CSSToXSLFOException

Constructs the filter with `debug` set to `false`.

public CSSToXSLFOFilter
(
java.net.URL baseUrl,
java.net.URL userAgentStyleSheet,
XMLReader parent
) throws CSSToXSLFOException

Constructs the filter with an empty `userAgentParameters` map and `debug` set to `false`.

public CSSToXSLFOFilter
(
java.net.URL baseUrl,
XMLReader parent
) throws CSSToXSLFOException

Constructs the filter with `userAgentStyleSheet` set to `null`, an empty `userAgentParameters` map and `debug` set to `false`.

public CSSToXSLFOFilter
(
XMLReader parent
) throws CSSToXSLFOException

Constructs the filter with `baseUrl` and `userAgentStyleSheet` set to `null`, an empty `userAgentParameters` map and `debug` set to `false`.

public CSSToXSLFOFilter
(
java.net.URL baseUrl,
java.net.URL userAgentStyleSheet,
java.util.Map userAgentParameters,
boolean debug
) throws CSSToXSLFOException

Constructs the filter without a parent.

public CSSToXSLFOFilter
(
java.net.URL baseUrl,
java.net.URL userAgentStyleSheet,
java.util.Map userAgentParameters
) throws CSSToXSLFOException

Constructs the filter without a parent and `debug` set to `false`.

public CSSToXSLFOFilter
(
java.net.URL baseUrl,
java.net.URL userAgentStyleSheet
) throws CSSToXSLFOException

Constructs the filter without a parent, an empty `userAgentParameters` map and `debug` set to `false`.

public CSSToXSLFOFilter(java.net.URL baseUrl) throws CSSToXSLFOException

Constructs the filter without a parent, `userAgentStyleSheet` set to `null`, an empty `userAgentParameters` map and `debug` set to `false`.

public CSSToXSLFOFilter() throws CSSToXSLFOException

Constructs the filter without a parent, `baseUrl` and `userAgentStyleSheet` set to `null`, an empty `userAgentParameters` map and `debug` set to `false`.

Methods

public java.net.URL getBaseUrl()

Returns the base URL of the filter. It is either set in a constructor or with the `setBaseUrl` method.

public java.util.Map getParameters()

Returns the User Agent parameters of the filter. They are either set in a constructor or with the `setUserAgentStyleSheet` method.

public java.net.URL getUserAgentStyleSheet()

Returns the User Agent style sheet of the filter. It is either set in a constructor or with the `setUserAgentStyleSheet` method.

public void setBaseUrl(java.net.URL baseUrl)

Sets the base URL of the filter. The base URL is used to resolve relative URLs in the input document. See also `baseUrl`.

public void setParameters(java.util.Map parameters)

Sets the User Agent parameters of the filter. See also `userAgentParameters`.

public void setUserAgentStyleSheet(java.net.URL userAgentStyleSheet)

Sets the User Agent style sheet of the filter. See also `userAgentStyleSheet`.

5.1.2 be.re.css.CSSToXSLFOException

This class extends java.lang.Exception.

Constructors

public CSSToXSLFOException(java.lang.Exception e)

This is just a wrapper around `e`.

public CSSToXSLFOException(java.lang.String message)

The parameter will be returned by the method `getMessage`.

5.1.3 be.re.css.CSSToXSLFO

This class contains a few convenience methods with which an XML filter set-up can be avoided, because they do it for you.

Methods

```
public static void convert
(
   java.io.InputStream in,
   java.io.OutputStream out,
   java.net.URL baseUrl,
   java.net.URL userAgentStyleSheet,
   java.net.URL catalog,
   java.util.Map userAgentParameters,
   java.net.URL[] preprocessors,
   boolean validate,
   boolean debug
) throws java.io.IOException, CSSToXSLFOException
```

in
: The input document. It must not be `null`.

out
: The output document. It must not be `null`.

baseUrl
: See `baseUrl`.

userAgentStyleSheet
: See `userAgentStyleSheet`.

catalog
: The catalog used to resolve entities during XML parsing. It must have the format defined by SGML Open Technical Resolution TR9401:1997. Only the "PUBLIC" and "SYSTEM" keywords are supported. It may be `null`.

userAgentParameters
: See `userAgentParameters`.

preprocessors
: An array of XSLT style sheets. The input goes through them in the specified order and before the `CSSToXSLFOFilter`.

validate
: Turns on validation during XML parsing of the input document.

debug
: See `debug`.

public static void convert
(
  java.io.InputStream in,
  java.io.OutputStream out,
  java.net.URL userAgentStyleSheet
) throws java.io.IOException, CSSToXSLFOException

Calls the first variant of `convert` with `baseUrl`, `catalog` and `preprocessors` set to `null`, `validate` and `debug` set to false and an empty `userAgentParameters` map.

public static void convert
(
  java.io.InputStream in,
  java.io.OutputStream out
) throws java.io.IOException, CSSToXSLFOException

Calls the first variant of `convert` with `baseUrl`, `userAgentStyleSheet`, `catalog` and `preprocessors` set to `null`, `validate` and `debug` set to false and an empty `userAgentParameters` map.

5.2 EXAMPLES

Since CSSToXSLFOFilter is derived from org.xml.sax.helpers.XMLFilterImpl., it implements all SAX event interfaces, as well as org.xml.sax.XMLFilter. As a consequence, the filter can occur in input and output filter chains.

5.2.1 Example 1

The most straight-forward scenario is an application that reads the input document from a file and that writes an XSL-FO document into another file. For this we need an XML parser that can produce SAX events. The parser implements the org.xml.sax.XMLReader interface, so we can make it the parent of CSSToXSLFOFilter.

In order to create a parser, we first have to set up the parser factory and make it namespace-aware. This happens at the lines 6 through 8. The filter can now be created with the input document as the base URL (in case any relative URLs need to be resolved) and an XML parser as its parent. This is done at the lines 9 through 14.

We now have to prepare the output part. We use an XSLT transformer without a style sheet to copy the SAX events to the output. The transformer must be in a form that accepts SAX events. This is why a javax.xml.transform.sax.TransformerHandler is created at lines 15 through 19. It implements the org.xml.sax.ContentHandler interface. By giving it the output file as a result (lines 20 through 26), the SAX events are transformed in the XML syntax.

The input and output parts can now be connected by setting the content handler of the filter to the transformer handler (line 27). The whole chain is then activated by calling the `parse` method, passing it the input document in the form of a file. The filter will pass this call onto the parser, which is its parent. The parser starts producing SAX events that go through the filter and into the transformer handler.

```
 1 public class Example1
 2 {
 3   public static void
 4   main(String[] args) throws Exception
 5   {
 6     javax.xml.parsers.SAXParserFactory factory =
 7       javax.xml.parsers.SAXParserFactory.newInstance();

 8     factory.setNamespaceAware(true);

 9     be.re.css.CSSToXSLFOFilter filter =
10       new be.re.css.CSSToXSLFOFilter
11       (
12         new java.io.File(args[0]).toURL(), // base URL.
13         factory.newSAXParser().getXMLReader()
14       );

15     javax.xml.transform.sax.TransformerHandler handler =
16       (
17         (javax.xml.transform.sax.SAXTransformerFactory)
18           javax.xml.transform.TransformerFactory.newInstance()
19       ).newTransformerHandler();

20     handler.setResult
21     (
22       new javax.xml.transform.stream.StreamResult
23       (
24         new java.io.File(args[1])
25       )
```

```
26      );

27      filter.setContentHandler(handler);

28      filter.parse
29      (
30        new org.xml.sax.InputSource
31        (
32          new java.io.FileInputStream(args[0])
33        )
34      );
35    }
36  }
```

5.2.2 Example 2

A variation of the previous example is to perform the transformation of the SAX events coming out of the filter to XML syntax in another way. In the previous example the parser had the control flow and the transformer acted as a handler of SAX events. We can also give the control flow to a transformer that reads the input and copies it to the output, because we don't give it any style sheet. We need to create a javax.xml.transform.Transformer. It is done at lines 15 through 17. The actual transformation is launched at lines 18 through 32. For this to work, we have to wrap our filter in a javax.xml.transform.sax.SAXSource. For the transformer it is as if it is going to call an XML parser.

```
 1 public class Example2
 2 {
 3   public static void
 4   main(String[] args) throws Exception
 5   {
 6     javax.xml.parsers.SAXParserFactory factory =
 7       javax.xml.parsers.SAXParserFactory.newInstance();

 8     factory.setNamespaceAware(true);

 9     be.re.css.CSSToXSLFOFilter filter =
10       new be.re.css.CSSToXSLFOFilter
11       (
12         new java.io.File(args[0]).toURL(), // base URL.
13         factory.newSAXParser().getXMLReader()
14       );

15     javax.xml.transform.Transformer transformer =
16       javax.xml.transform.TransformerFactory.newInstance().
17         newTransformer();

18     transformer.transform
19     (
20       new javax.xml.transform.sax.SAXSource
21       (
22         filter, // Acts as the XMLReader.
23         new org.xml.sax.InputSource
```

```
24          (
25            new java.io.FileInputStream(args[0])
26          )
27        ),
28        new javax.xml.transform.stream.StreamResult
29        (
30          new java.io.File(args[1])
31        )
32      );
33    }
34  }
```

5.2.3 Example 3

This example shows how a pre-processing step can be added to the filter chain. The input document is transformed by the pre-processor and the resulting SAX events go through the conversion filter. The pre-processor is created at lines 9 through 13. This one does nothing, i.e. it lets the events go through unmodified. In reality you would replace it with a class of your own.

The pre-processor instead of the filter is now initialised with the XML parser as its parent. The pre-processor will become the parent of the filter, as shown at line 18. When the `parse` method is called, the filter passes the call onto the pre-processor, which in turn passes it onto the parser. The SAX events produced by the parser will then flow through the pre-processor, which in turn forwards them, possibly modified, to the filter.

```
 1 public class Example3
 2 {
 3   public static void
 4   main(String[] args) throws Exception
 5   {
 6     javax.xml.parsers.SAXParserFactory factory =
 7       javax.xml.parsers.SAXParserFactory.newInstance();

 8     factory.setNamespaceAware(true);

 9     org.xml.sax.helpers.XMLFilterImpl myPreprocessor =
10       new org.xml.sax.helpers.XMLFilterImpl
11       (
12         factory.newSAXParser().getXMLReader()
13       );

14     be.re.css.CSSToXSLFOFilter filter =
15       new be.re.css.CSSToXSLFOFilter
16       (
17         new java.io.File(args[0]).toURL(), // base URL.
18         myPreprocessor
19       );

20     javax.xml.transform.sax.TransformerHandler handler =
21       (
22         (javax.xml.transform.sax.SAXTransformerFactory)
23           javax.xml.transform.TransformerFactory.newInstance()
```

```
24       ).newTransformerHandler();

25     handler.setResult
26     (
27       new javax.xml.transform.stream.StreamResult
28       (
29         new java.io.File(args[1])
30       )
31     );

32     filter.setContentHandler(handler);

33     filter.parse
34     (
35       new org.xml.sax.InputSource
36       (
37         new java.io.FileInputStream(args[0])
38       )
39     );
40   }
41 }
```

5.2.4 Example 4

The previous example can be modified in such a way that the pre-processor is an XSLT style sheet. From this style sheet a org.xml.sax.XMLFilter must be made, because it will sit between the XML parser and the filter. This is shown at lines 9 through 19. The transformer factory is re-used afterwards to create also the output handler.

```
 1 public class Example4
 2 {
 3   public static void
 4   main(String[] args) throws Exception
 5   {
 6     javax.xml.parsers.SAXParserFactory factory =
 7       javax.xml.parsers.SAXParserFactory.newInstance();

 8     factory.setNamespaceAware(true);

 9     javax.xml.transform.sax.SAXTransformerFactory trFactory =
10       (javax.xml.transform.sax.SAXTransformerFactory)
11         javax.xml.transform.TransformerFactory.newInstance();

12     org.xml.sax.XMLFilter myPreprocessor =
13       trFactory.newXMLFilter
14       (
15         new javax.xml.transform.stream.StreamSource
16         (
17           new java.io.File(args[2])
18         )
19       );

20     myPreprocessor.setParent
21     (
```

```
22        factory.newSAXParser().getXMLReader()
23      );

24      be.re.css.CSSToXSLFOFilter filter =
25        new be.re.css.CSSToXSLFOFilter
26        (
27          new java.io.File(args[0]).toURL(), // base URL.
28          myPreprocessor
29        );

30      javax.xml.transform.sax.TransformerHandler handler =
31        trFactory.newTransformerHandler();

32      handler.setResult
33      (
34        new javax.xml.transform.stream.StreamResult
35        (
36          new java.io.File(args[1])
37        )
38      );

39      filter.setContentHandler(handler);

40      filter.parse
41      (
42        new org.xml.sax.InputSource
43        (
44          new java.io.FileInputStream(args[0])
45        )
46      );
47    }
48  }
```

5.2.5 Example 5

In all previous examples we have been parsing an input document. In some applications, however, the data might come from somewhere else. It is possible, for example, to synthesize the XML from data that resides in the database. In such a scenario our filter no longer has a parent but becomes the SAX event handler of some system method, `generateReport` in this example.[1] This system method has the control flow. It fetches the data and generates the SAX events. In the case the generated XML stream is not suitable for CSS conversion, a pre-processor may be specified as the parent of the filter.

```
1 public class Example5
2 {
3   public static void
4   main(String[] args) throws Exception
5   {
6     be.re.css.CSSToXSLFOFilter filter =
```

1 Note that a real system method would probably need more than just the filter to do its work. It would therefore have more parameters.

```
 7       new be.re.css.CSSToXSLFOFilter();

 8     javax.xml.transform.sax.TransformerHandler handler =
 9       (
10         (javax.xml.transform.sax.SAXTransformerFactory)
11           javax.xml.transform.TransformerFactory.newInstance()
12       ).newTransformerHandler();

13     handler.setResult
14     (
15       new javax.xml.transform.stream.StreamResult
16       (
17         new java.io.File(args[0])
18       )
19     );

20     filter.setContentHandler(handler);
21     generateReport(filter);
22   }

23   private static void
24   generateReport(org.xml.sax.ContentHandler handler)
25   {
26   }
27 }
```

5.2.6 Example 6

It may be the case that you want to synthesize the XML stream in a system method, which needs the control flow, but that the interface of your XSL-FO formatter is such that it also needs the control flow. In other words, the formatter is not available in the form of a SAX event handler, but has some method that must be called to perform the actual formatting. At lines 21 through 31 there a hypothetical example of such a formatter.

To solve this control flow conflict you can create an adapter that implements the org.xml.sax.XMLReader interface. Instead of actually parsing some XML you let both `parse` methods call your system method. The parameters the latter needs are passed through the constructor of the adapter. When the formatter now calls the `parse` method it really ends up calling the system method, which synthesizes the SAX events.

```
 1 public class Example6
 2 {
 3   public static void
 4   main(String[] args) throws Exception
 5   {
 6     be.re.css.CSSToXSLFOFilter filter =
 7       new be.re.css.CSSToXSLFOFilter
 8       (
 9         new MyReportGenerator(new Object())
10       );
11     MyXSLFOFormatter  myFormatter = new MyXSLFOFormatter();

12     myFormatter.format
```

```
13      (
14        new javax.xml.transform.sax.SAXSource(filter, null),
15        new java.io.FileOutputStream(args[0]));
16    }

17    private static void
18    generateReport(Object context)
19    {
20    }

21    public static class MyXSLFOFormatter
22    {
23      public void
24      format
25      (
26        javax.xml.transform.Source source,
27        java.io.OutputStream out
28      )
29      {
30      }
31    }

32    public static class MyReportGenerator
33      extends org.xml.sax.helpers.XMLFilterImpl
34    {
35      private Object context;

36      public
37      MyReportGenerator(Object context)
38      {
39        this.context = context;
40      }

41      public void
42      parse(org.xml.sax.InputSource input)
43        throws org.xml.sax.SAXException, java.io.IOException
44      {
45        generateReport(context);
46      }

47      public void
48      parse(String systemId)
49        throws org.xml.sax.SAXException, java.io.IOException
50      {
51        generateReport(context);
52      }
53    }
54  }
```

SOME TECHNIQUES 6

A few practical cases of formatting contructs, which are either more advanced or not yet very common, are described in this chapter. Gradually, new cases will be added. The chapter is some sort of "how to" section in the user guide. The examples use XHTML as the input document language.

6.1 CUSTOMISING LIST LABELS WITH MARKERS

The genaration of the labels of an itemised list is somewhat fixed. It depends on the value of the `list-style-type` property.[1] Sometimes, however, more control is required over how the labels look like. This can be achieved through markers.

Basically, you have to specify a `:before` pseudo element with the display type `marker` in your style sheet for those elements you have given the display type `list-item`. Strictly speaking that display type is not needed, but if you are about to convert your existing lists, those elements would have that display type.

In the pseudo element you have control over the formatting of the label. The only exception is that the `width` property must be fixed. The tool doesn't support the automatic calculation of the required width. If your style sheet doesn't specify a width, a default value will be used. In order to not depend on this value, it is best to specify one.

The following example is an ordered list with a nested ordered list in the second item. We are going to change the numbering as well as the alignment of the labels.

```
<ol>
  <li>Item 1</li>
  <li>Item 2
    <ol>
      <li>Subitem 1</li>
      <li>Subitem 2</li>
      <li>Subitem 3</li>
    </ol>
  </li>
  <li>Item 3</li>
</ol>
```

In the style sheet we say that the `:before` pseudo element of any `li` under a `ol`, no matter the level, is a marker. In there, we increment the counter that is reset for each level of `ol`. We also display it with the `lower-roman` counter style instead of the default style (`decimal`). This style will show the effect of the right alignment of

1 The `list-style-image` and `list-style-position` properties are not supported by this tool.

the text inside the label. The `marker-offset` property provides for a bit of space between the label and the list item body.

The `width` property deserves special attention. First of all it defines the width of the labels. Since markers shouldn't influence the positioning of the element they are attached to, the labels would stick out to the left by the amount of the value of the `width` property. In order to compensate this, we have to add a `margin-left` with the same value to the list item itself.

```
ol { counter-reset: list-counter; }

ol li { margin-left: 2em; }

ol li:before
{
  content: counter(list-counter, lower-roman) ".";
  counter-increment: list-counter;
  display: marker;
  marker-offset: 0.5em;
  text-align: right;
  width: 2em;
}
```

The rendered result would like this:

i. Item 1

ii. Item 2

    i. Subitem 1

    ii. Subitem 2

    iii. Subitem 3

iii. Item 3

6.2 MAKING SECTION NUMBERS "STICK OUT"

Sometimes the text of the section titles must be aligned with the rest of the material, at the left side for example. As consequence, if the titles also have section numbers, those will stick out at the left side of the title, into the margin, just like the title of the current section. This can be obtained by specifying a `:before` pseudo element for the section titles with the display type `marker`. Because markers shouldn't influence the positioning of their associated element, the marker content is prepended. This is the piece of style sheet you would need:

```
h2:before
{
  display: marker;
  marker-offset: 0.5em;
  padding-right: 0pt;
  text-align: right;
```

```
  width: 3em;
}
```

6.3 THIS GUIDE'S PAGE SET-UP

The page set-up of this guide is rather advanced and is therefore an interesting practical case. The difficulty lies in specifying the static regions if there are many kinds of pages and if for each of those the static regions are different. i.e. very specific.

In order to avoid an explosion of CSS property specifications for all those regions, we can work in a sort of multidimensional way. This is possible through the combination of two things. First, we have the CSS cascading mechanism, which allows us to centralise common property specifications. Second, the can make use of the fact that the `class` attribute is a space-separated list of names. Because of this, an element can belong to several classes. These are the static regions of this guide:

```
<div class="first top"/>
<div class="title first bottom"/>
<div class="title first top"/>
<div class="copy-right first bottom"/>
<div class="copy-right first top"/>
<div class="left bottom"><span/></div>
<div class="right bottom"><span/></div>
<div class="left top"><span/></div>
<div class="right top"><span/></div>
<div class="front left bottom"><span/></div>
<div class="front right bottom"><span/></div>
<div class="blank bottom"/>
<div class="blank top"/>
```

Neither of them has a lot of content. It is all generated using the `span` element as a hook. For the empty `div` elements no content will be generated. You can see that all regions belong to more than one class. The more classes they belong to, the more specific is the set of pages they apply to. This is one dimension. The other is whether the static region is a top or bottom region.

The first region, for example, is a bottom region that applies only to the first page of the named page sequence called "title". This is expressed in the style sheet as follows:

```
div.title.first
{
  page: first-title;
}
```

The rule applies if `class` is "title" *and* "first". This is more specific then the following page assignment which also occurs in the style sheet:

```
div.first
{
  page: first;
}
```

Another example is the pair of bottom regions for left and right pages. They apply for left and right pages, no matter the page sequence. For the “front” page sequence, however, there is a more specific version. Making it more specific is done by the two page assignments in the following style sheet part. The style of the bottom region of the front part is a bit different. It display the page numbers in lower Roman. The complete style is obtained by cascading all the rules for the class “bottom”. The first two rules define the style for all bottom regions and the last one overrides the counter style.

```
div.bottom
{
  height: 3em;
  padding-top: 2em;
  region: bottom;
}

div.bottom > span:before
{
  content: counter(page);
}

div.front.left
{
  page: left-front;
}

div.front.right
{
  page: right-front;
}

div.front.bottom > span:before
{
  content: counter(page, lower-roman);
}
```

For blank pages we want no static regions at all, at least not visually. The presence of the top and bottom regions is defined generally through the `div.top` and `div.bottom` rules. As a consequence, no matter the page sequence, those static regions are always generated. All we have to do now is making sure they don't contain anything. This is done by the two empty regions with the `class` attribute set to “blank top” and “blank bottom” respectively. Those regions are assigned to blank pages with the following style sheet part:

```
div.blank
{
  page: blank;
}
```

There is one more special construct left to discuss: the absence of static top regions on the first page of a chapter. As with blank pages they are not really absent. They are merely made empty. It is in fact the first region with the classes “first” and “top”. This

region is assigned to the `first` pseudo page. All chapters are however in the named page sequence "main". If we do nothing only the first page of the first chapter will have an empty top region. We therefore should toggle the `page` property without adding extra pages. This can be achieved by inserting an empty `div` element between the chapters with the class "separator". The page assignment for that class is "separator". This named page is not used for anything else. Since the element is empty no page sequence is generated. The next "main" element, however, will start a new page sequence.

After the front matter not only the page number style changes to decimal, but the page numbering is also reset. We can't just reset the `page` counter in the "main" page context, because then the numbering would be reset for each chapter. Instead, the style sheet defines a `main-first` @page rule, which contains the same definitions as `main` and a `page` counter reset on top of it. This page is assigned to the first chapter only, using the `first-child` pseudo-class on the `h1` element.

6.4 A TWO-COLUMN ARTICLE

Many articles and papers are formatted in two column mode. The title, abstract, authors, etc. are usually displayed across the two columns. With two extension properties it is possible to do this. The `column-count` should be set to "2" in the page context. The title material can be wrapped in a block element for which the `column-span` property is set to "all".

6.5 INITIAL CAPITALS

A typographical effect that is often used are initial capitals. It consists of making the first letter of an article or chapter stand out by rendering it bigger and perhaps in another font and/or colour. In CSS this is supported through the `:first-letter` pseudo element, which is described in section 5.12.2 of [CSS2]. In CSSTOXSLFO it is implemented with the restriction that letter combinations, which are considered as one letter, are not examined. In case you need that, you can always use the Unicode ligature characters instead.

The technique was applied to the previous paragraph using the piece of style sheet below. Note the second deviation from the specification being the usage of the property `vertical-align` while the `float` property has the value `none`. It is allowed in CSSTOXSLFO because otherwise we have no control over the alignment of the first letter with the lines next to it. This depends on the font and will always require some trial and error in order to get it right. The values for the other properties are obtained in the same way. In fact, for this special case, we work around the normal way a glyph is layed out in a line.

```
p:first-letter
{
  font-family: serif-swash;
  font-size: 46pt;
  font-style: italic;
  float: left;
```

```
    line-height: 46pt;
    padding-right: 6pt;
    margin-bottom: -12pt;
    vertical-align: 9pt;
  }
```

SPECIAL PROVISIONS FOR XHTML

A

While the tool works for any XML vocabulary it does a number of things for XHTML specifically. Other vocabularies may be supported in the same way at some later stage. The items are the following:

- Non-CSS presentational hints are translated to the corresponding CSS rules, as prescribed in section 6.4.4 of [CSS2];
- The `lang` attribute is honored;
- Hyperlinks are recognized and translated in XSL-FO links;
- The `link` element can be used to specify external style sheets;
- Style sheets can be embedded with the `style` element;
- The `style` attribute is honored;
- The `img` element is interpreted and processed;
- The `html-header-mark` user agent parameter is available;
- There is a user agent style sheet for XHTML that cascades against the one in appendix A of [CSS2].

THE USER AGENT STYLE SHEET

B

B.1 XHTML

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

@media print
{
  a[href]
  {
    color: blue;
    link: attr(href);
    text-decoration: none;
  }

  a[name]
  {
    anchor: name;
  }

  blockquote, dl, ol, p, ul
  {
    margin: 0.83em 0pt;
  }

  blockquote
  {
    margin-left: 3em;
    margin-right: 3em;
  }

  body
  {
    font-family: serif;
    padding: 0pt;
    region: body;
  }

  body:lang(da)
  {
    quotes: "\00BB" "\00AB";
  }

  body:lang(de-DE), body:lang(de-AT)
  {
    quotes: "\201E" "\201C" "\201A" "\2018"
```

```
}

body, body:lang(en), body:lang(es)
{
  quotes: "\201C" "\201D" "\2018" "\2019";
}

body:lang(fr)
{
  quotes: "\00AB " " \00BB" "\2039 " " \203A";
}

body:lang(it)
{
  quotes: "\00AB " " \00BB";
}

body:lang(nl)
{
  quotes: "\201D" "\201D" "\2019" "\2019";
}

body:lang(no), bodylang:(pt), body:lang(de-CH)
{
  quotes: "\00AB" "\00BB" "\2039" "\203A"
}

body:lang(sv)
{
  quotes: "\00BB" "\00BB";
}

caption
{
  margin: 0.5em 0pt;
}

dt
{
  page-break-after: avoid;
}

h1
{
  font-size: 1.6em;
  margin-bottom: 0.7em;
  margin-top: 1.4em;
}

h2
{
  font-size: 1.3em;
  margin-bottom: 0.6em;
  margin-top: 1.2em;
}
```

```
h3
{
  font-size: 1.1em;
}

h3, h4
{
  margin-bottom: 0.5em;
  margin-top: 1em;
}

h1, h2, h3, h4, h5, h6
{
  hyphenate: false;
}

hr
{
  border: 0.1pt solid;
}

img
{
  content-height: scale-to-fit;
  content-width: scale-to-fit;
  display: graphic;
  scaling: uniform;
  src: attr(src);
}

li
{
  margin-bottom: 0.8em;
  margin-top: 0.8em;
}

li p, li blockquote, li dl, li ol, li ul
{
  margin-bottom: 0.5em;
  margin-top: 0.5em;
}

li li
{
  margin-bottom: 0.5em;
  margin-top: 0.5em;
}

li li p, li li blockquote, li li dl, li li ol, li li ul
{
  margin-bottom: 0.3em;
  margin-top: 0.3em;
}
```

```
li li li
{
  margin-bottom: 0.4em;
  margin-top: 0.4em;
}

li li li p, li li li blockquote, li li li dl, li li li ol,
  li li li ul
{
  margin-bottom: 0.3em;
  margin-top: 0.3em;
}

li, p
{
  text-align: justify;
}

pre
{
  font-size: 0.85em;
}

ul
{
  list-style-type: disc;
}

ol li ul, ul li ul
{
  list-style-type: circle;
}

ol li ol li ul, ol li ul li ul, ul li ol li ul, ul li ul li ul
{
  list-style-type: square;
}

q:after
{
  content: close-quote;
}

q:before
{
  content: open-quote;
}

script
{
  display: none;
}

span.section-number
{
```

```
    padding-right: 1em;
  }
}
```

B.2 DELTAXML

```
@namespace deltaxml
  url(http://www.deltaxml.com/ns/well-formed-delta-v1);

@media print
{
  deltaxml|PCDATAnew, deltaxml|PCDATAold
  {
    display: inline;
  }

  deltaxml|exchange, deltaxml|new, deltaxml|old
  {
    display: wrapper;
  }

  *[deltaxml|delta="add"], deltaxml|PCDATAnew, deltaxml|new
  {
    text-decoration: underline;
  }

  *[deltaxml|delta="delete"], deltaxml|PCDATAold, deltaxml|old
  {
    text-decoration: line-through;
  }

  *[deltaxml|delta="add"]:before, deltaxml|PCDATAnew:before,
    deltaxml|new, *[deltaxml|delta="delete"]:before,
    deltaxml|PCDATAold:before, deltaxml|old
  {
    change-bar-class: changed;
    change-bar-placement: alternate;
    change-bar-style: solid;
    change-bar-width: 0.2pt;
  }

  *[deltaxml|delta="add"]:after, deltaxml|PCDATAnew:after,
    *[deltaxml|delta="delete"]:after, deltaxml|PCDATAold:after
  {
    change-bar-class: changed;
  }
}
```

B.3 XLINK

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

@media print
{
  *[xlink|href]
  {
    link: attr(xlink|href);
  }
}
```

REFERENCES

C

[CSS2]
"Cascading Style Sheets, level 2, CSS2 Specification", W3C Recommendation 12 May 1998, Bert Bos, Håkon Wium Lie, Chris Lilley, Ian Jacobs, http://www.w3.org/TR/1998/REC-CSS2-19980512.

[CSS3G]
"CSS3 Generated and Replaced Content Module", W3C Working Draft 14 May 2003, Ian Hickson, http://www.w3.org/TR/2003/WD-css3-content-20030514.

[CSS3L]
"CSS3 module: Lists", W3C Working Draft 7 November 2002, Ian Hickson, Tantek Çelik, http://www.w3.org/TR/2004/WD-css3-lists-20021107.

[CSS3P]
"CSS3 Paged Media Module", W3C Candidate Recommendation 25 February 2004, Håkon Wium Lie, Jim Bigelow, http://www.w3.org/TR/2004/CR-css3-page-20040225.

[CSS3S]
"CSS3 Selectors", W3C Candidate Recommendation 13 November 2001, Daniel Glazman, Tantek Çelik, Ian Hickson, http://www.w3.org/TR/2001/CR-css3-selectors-20011113.

[DELTA]
"How DeltaXML Represents Changes to XML Files", DeltaXML Ltd., http://www.deltaxml.com/library/how-deltaxml-represents-changes.html.

[NAMES]
"Namespaces in XML", W3C Recommendation 14 January 1999, Tim Bray, Dave Hollander, Andrew Layman, http://www.w3.org/TR/REC-xml-names/.

[XHTML]
"XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition)", W3C Recommendation 26 January 2000, revised 1 August 2002, http://www.w3.org/TR/2002/REC-xhtml1-20020801.

[XLINK]
"XML Linking Language (XLink) Version 1.0", W3C Recommendation 27 June 2001, Steve DeRose, Eve Maler, David Orchard, http://www.w3.org/TR/2001/REC-xlink-20010627.

[XSL-FO]
"Extensible Stylesheet Language (XSL), Version 1.0", W3C Recommendation 15 October 2001, Sharon Adler, Anders Berglund, Jeff Caruso, Stephen Deach, Tony Graham, Paul Grosso, Eduardo Gutentag, Alex Milowski, Scott Parnell, Jeremy Richman, Steve Zilles, http://www.w3.org/TR/2001/REC-xsl-20011015/.

[XSL-FO11]
"Extensible Stylesheet Language (XSL), Version 1.1", W3C Working Draft 16 December 2004, Anders Berglund, http://www.w3.org/TR/2004/WD-xsl11-20041216/.