How to Develop Stylesheets for XML to XSL-FO Transformation

June, 2001

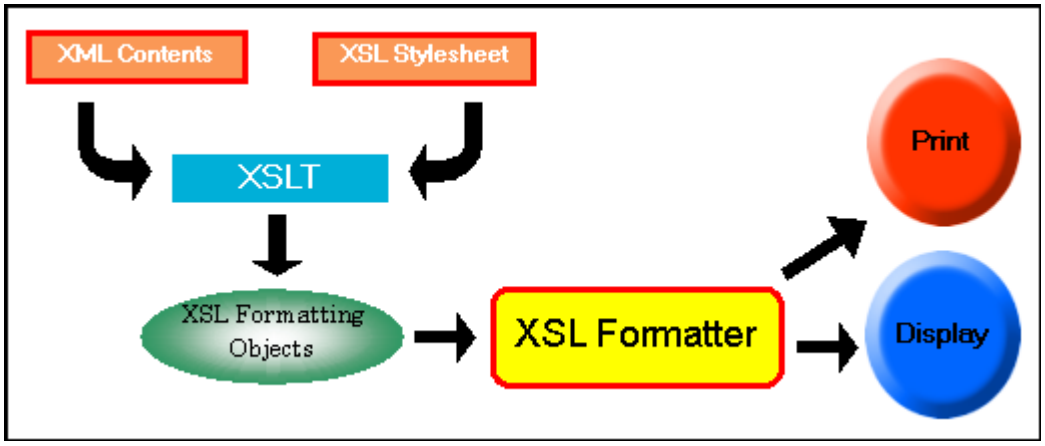
Antenna House, Inc.

Table of Contents

Preface

XSL has been brought to the attention of a wide audience as a specification for displaying and printing the XML document. The following is general process to transform the XML document into XSL Formattiong Objects (XSL-FO) and print it.

- Develop stylesheets that conforms to the DTD of source XML document to create the target output.
- Input the XML document and the XSLT stylesheet to the XSLT processor to create XSL-FO.
- Get the target outputs such as prints, PDF outputs, by the XSL-FO processor.



Generating XSL-FO and Display/Print by XSL Formatter

The knowledge about XSLT and XSL is necessary to develop XSLT stylesheets. The XSL specification has a huge amount of contents, it has over 400 pages. It is pretty hard to understand this specification. But basically it is intended for implementers. It is not necessary for XSLT stylesheet designers to understand everything. You can fully write stylesheets by knowing some regular contents and patterns.

How is the developing environment of making stylesheets? For the present, we have no means to edit XSLT stylesheets for XSL in GUI mode. Actually, we have to edit stylesheets manually by using an editor basically. After editing stylesheets, the target output can not be gained at one time. It often happens to debug the transformation process by XSLT processor. In this field, some tools began to appear. For further development of technology, it is quite sure that the developing environment and tools concerning XSLT and XSL-FO will be much more improved. But so far, it is important to get experience about developing stylesheets.

This report explains how to edit stylesheets which are used for transforming XML documents into XSL-FO according to the example of SampleDoc. SampleDoc is made for this report as a format to write a sample document. This is based on PureSmartDoc presented by Tomoharu Asami. To make it a sample, the number of the elements is reduced and the usuful functions for writing and formatting documents are added. SampleDoc has almost the same grammer as HTML and the logic of the documents succeeds the functions of PureSmartDoc inherited from LaTex.

Step for XSL-FO Transformation

Now, what steps are necessary to develop XSLT stylesheets? These steps are explained briefly as below.

Steps	Contents
Know the structure of the XML document	First, the information about the structure of XML source documents is required. XSLT processor can transform XML document into XSL-FO without a DTD. But the information described in the DTD such as kind of elements, content of elements, appearing order of elements and value of attributes are quite necessary in order to develop a stylesheet.
Specify a printing form	This is the printing form as a final output, in other words the output specification. XSL is a formatting specification. Printing forms has various range of specifications such as size and layouts of printing paper, layouts of head and body, deciding whether or not to output index, table of contents, and so on.
Apply a printing form to formatting objects	After determining the specification of printing, you have to know what XSL formatting objects and properties are applied in order to print in this style. It is better to practice how to specify by referring to a sample stylesheet.
Develop a XSLT stylesheet	Put the instructions to the stylesheet in order to transfrom XML source documents into the target printing form. Map the XML source document to XSL formatting objects that can generate the output specification. The stylesheets have the similar aspect as the general programing languages, while it may be difficult if you do not understand the feature of the XSLT. (1)

(1) Refer to the definition list template in this report. In XSLT, Structure for control of the conditional branch can be made, but it is impossible to assign a value to a variable. The technique to realize by calling loops recursively is necessary.

The Structure of SampleDoc

The following table shows the structure of SampleDoc treated in this report. For more detail, refer to SampleDoc. dtd

Element	Meaning	Definition
a group of block elements	—	p | figure | ul | ol | dl | table | program | div
a group of inline elements	—	a | note | span | b | i | em | code | br
doc	root element	(head, body)
head	header	(date | author | abstract | title)*
date, author, abstract, title	header elements, date, author, abstract, title	(#PCDATA | a group of inline elements)*
body	body	(chapter | part | section | a group of block elements | a group of inline elements)*
part	part	(title, (chapter |a group of block elements | a group of inline elements) *)
chapter	chapter	(title, (section | a group of bock elements | a group of inline elements)*)
section	section	(title, (subsection | a group of block elements | a group of inline elements)*)
subsection	subsection	(title, (subsubsection | a group of block elements | a group of inline elements)*)
subsubsection	subsubsection	(title, (a group of block elements | a group of inline elements)*)
title	title	(#PCDATA | a group of inline elements)*
p	paragraph	(#PCDATA | a group of block elements | a group of inline elements)*
figure	figure	(title?) Specify a file by the src attribute.
ul	unordered list	(li*) specify a character for label of line by the type attribute.
ol	ordered list	(li*) Specify format of number in the label by the type attribute.
dl	definition list	(dt, dd)* Specify whether to format the block in horizontal way or in vertical way by the type attribute.
dt	definition term	(#PCDATA | a group of block elements | a group of inline elements)*
dd	description of details	(#PCDATA | a group of block elements | a group of inline elements)*
table	entire table	(title?, col*, thead?, tfoot?, tbody) Specify whether to make auto layout or fixed by the layout attribute. Specify the width of the entire table by the width attribute. Specify the row height of the entire table by the rowheight attribute.
col	column format	EMPTY Speciry the number of the column by the number attribute, the width of the column by the width attribute.
thead	table header	(tr*)

Element	Meaning	Definition
tfoot	table footer	(tr*)
tbody	table body	(tr*)
tr	table row	(th | td)* Specify the height of the row by the height attribute.
th	table header	(#PCDATA | a group of inline elements | a group of block elements)* Specify the number of the columns to be expanded accross,the number of the rows to be expanded vertically by the colspan and rowspan attributes. The align attribute allows horizontal alignment to be set to left, right, or center. The valign attribte allows vertical alignment to be set to top, middle, bottom or baseline.
td	table data	(#PCDATA | a group of inline elements | a group of block elements)* Specify the number of the columns to be expanded accross,the number of the rows to be expanded vertically by the colspan and rowspan attributes. The align attribute allows horizontal alignment to be set to left, right, or center. The valign attribte allows vertical alignment to be set to top, middle, bottom or baseline.
program	program code	(#PCDATA | title)*
div	general block element	(title, (general block element | general inline element)*) The div element expands the type by the class attribute.
a	anchor(link)	(#PCDATA | a group of inline elements)* Specify URL as the value of href attribute.
note	note	(#PCDATA | a group of inline elements)*
b	bold typeface	(#PCDATA | a group of inline elements)*
i	italic typeface	(#PCDATA | a group of inline elements)*
em	emphasis	(#PCDATA | a group of inline elements)*
code	program code of the in-line elements	(#PCDATA | a group of inline elements)*
span	general in-line element	(#PCDATA | a group of inline elements)*
br	line break	EMPTY

The feature of SampleDoc is as follows:

- The document structure starting from part to subsubsection is the same as that of PureSampleDoc. You can start writing from part, also start from section. It has a flexible sturucture so that it can map to various kinds of documents.
- In SampleDoc, the number of the block element and in-line element are reduced to a minimun amount compared to PureSmartDoc. The div in the block element and the span in the in-line element are added to give various extentions.
- The br element is added so that you can break lines inside of the lists, or the cells in the table, also inside of the paragraph without ending the paragraph.
- Output format of the list and table can be specified by the attributes. [(2)]

[(2)]The problem is that the content and style is mixed.

Hello! World

A Simple Example of Transforming SampleDoc into XSL-FO

Now, we show the simplest sample XSLT stylesheet[3] that transforms SampleDoc to XSL-FO as follows.

Source XML Document（Hello.xml)

```
<?xml version="1.0" encoding="UTF-16" ?>
<doc>
  <head>
    <title>Sample</title>
  </head>
  <body>
    <p>Hello World!</p>
    <p>This is the first<b>SampleDoc</b></p>
  </body>
</doc>
```

XSLT Stylesheet(Sample.xsl)

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0" xmlns:fo="http://www.w3.org/1999/XSL/Format"
                              xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" version="1.0" indent="yes" />

<xsl:template match="doc">
  <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
    <fo:layout-master-set>
      <fo:simple-page-master page-height="297mm" page-width="210mm"
          margin="5mm 25mm 5mm 25mm" master-name="PageMaster">
        <fo:region-body   margin="20mm 0mm 20mm 0mm"/>
      </fo:simple-page-master>
    </fo:layout-master-set>
    <fo:page-sequence master-name="PageMaster">
      <fo:flow flow-name="xsl-region-body" >
        <fo:block>
          <xsl:apply-templates select="body"/>
        </fo:block>
      </fo:flow>
    </fo:page-sequence>
  </fo:root>
</xsl:template>

<xsl:template match="body">
  <xsl:apply-templates />
</xsl:template>

<xsl:template match="p">
  <fo:block>
    <xsl:apply-templates />
  </fo:block>
</xsl:template>
```

[3]The sample XSLT stylesheet in this document conforms to XSL Candidate Recommendation. We plan to revise this document in the near future in accordance with XSL Formatter V2.0.

```
<xsl:template match="b">
  <fo:inline font-weight="bold">
    <xsl:apply-templates />
  </fo:inline>
</xsl:template>

</xsl:stylesheet >
```

Generated XSL-FO

```
<?xml version="1.0" encoding="UTF-16"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-master-set>
    <fo:simple-page-master page-height="297mm" page-width="210mm"
          margin="5mm 25mm 5mm 25mm" master-name="PageMaster">
      <fo:region-body margin="20mm 0mm 20mm 0mm"/>
    </fo:simple-page-master>
  </fo:layout-master-set>
  <fo:page-sequence master-name="PageMaster">
    <fo:flow flow-name="xsl-region-body">
      <fo:block>
        <fo:block>Hello World!</fo:block>
        <fo:block>This is the first
          <fo:inline font-weight="bold">SampleDoc</fo:inline>
        </fo:block>
      </fo:block>
    </fo:flow>
  </fo:page-sequence>
</fo:root>
```

The above XSL-FO is formatted/displayed as follows. [4]

Hello World!
This is the first **SampleDoc**

Stylesheet Structure

The above Sample.xsl and XSL-FO show the following facts:

- Most of the stylesheet consinsts of templates. The descendants of the root element consists of xsl:template elements. Each template (xsl:template) specifies the transformation that is to be applied to a element of the source XML document by the instruction match="xxx"
- Formatting objects and the XML source text in each template are output to result XSL-FO tree. Then, templates that match to the descendant elements are called by an instruction of xsl:apply-templates.

XSLT processor loads the source XML document, starts processing from the root node. It finds the templates that match each node, and processes them as described in the templates. The processor processes child elements one after another, continues untill the processor returns to the root element and there are no more templates to be processed.

[4]Our product XSL Formatter can save the output of XSLT processor to a file in the formatter option.

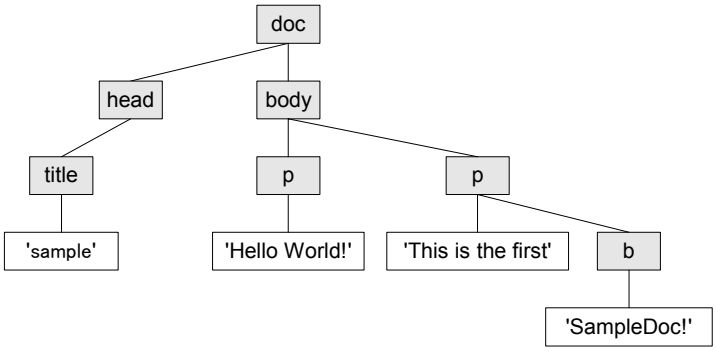
Block Element and Inline Element

Please note how the XSLT stylesheet maps block elements and inline elements in source element to formatting objects.

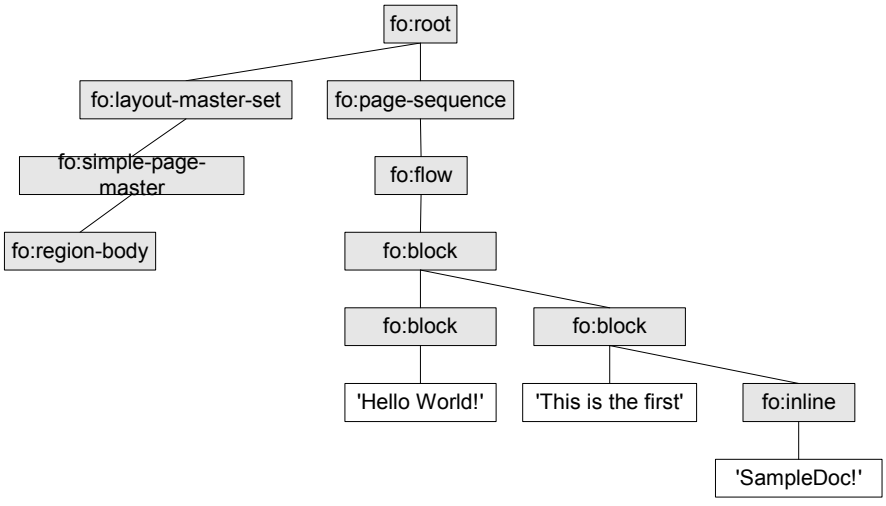
- In the stylesheet, p elements are transformed into fo:block objects, b elements are transformed into fo:inline objects. The base of XSL-FO transformation is to map the elements of the source XML documents to either fo: block elements or fo:inline elements according to the layout instruction.
- The elements that intend to break lines by the end tag normaly can be mapped to the fo:block objects. the elements of which the end tag do not intend to break lines can be mapped to the fo:inline objects. The attributes of source elements specify properties of formatting objects. In this case, the b element means emphasis and property of the inline object generated from b is specified as bold.

XSL-FO Tree Structure

Next, please pay attention to the XSL-FO tree structure. The root of the XSL-FO tree is fo:root, which has two children, fo:layout-master-set and fo:page-sequence. Fo:layout-master-set defines the page layouts and fo:page-sequence has a flow of contents to be set in pages.



Hello.xml Tree Structure



XSL-FO Tree after XSLT Processing

Fo:layout-master-set that defines the page layouts should precede fo:page-sequence that is an actual content of pages. XSLT processor processes the XML source document from the root element, seeking the templates (xsl: template) to be matched. Therefore, **fo:layout-master-set element should be an output from the template that processes the root element of the XML source document.** In this case, <xsl:template match="doc"> takes this processing.

The following node of the fo:flow is the same tree structure as the original document though the element names has changed. **The nodes existed in the original document**are transfered as it used to be, using <xsl:template match="xxx">～<xsl:apply-templates;> The result has the same tree structure. In the Sample.xsl, the information described in <head>～</head> of source XML document is not output. It is because the child element <body>is specified to be applied but <head> element is avoided according to the instruction of <xsl:apply-templates select="body" />

Printing Form Specification

We can get only a simple output when an XML document is processed according to the sample in the previous chapter. In order to get more rich outputs, we add the following specification.

【 Page Format 】

Item	Specification
Paper size	Letter size (8.5in x 11in)
Orientation	Portrait
Writing mode	lr-tb
Margins	Margin Top：5mm, Bottom：5mm, Left：25mm, Right：25mm
Header, Hooter	Specify the header, footer regions. Do not set text inside the whitespace on the both side of the body.

【 Header Region 】

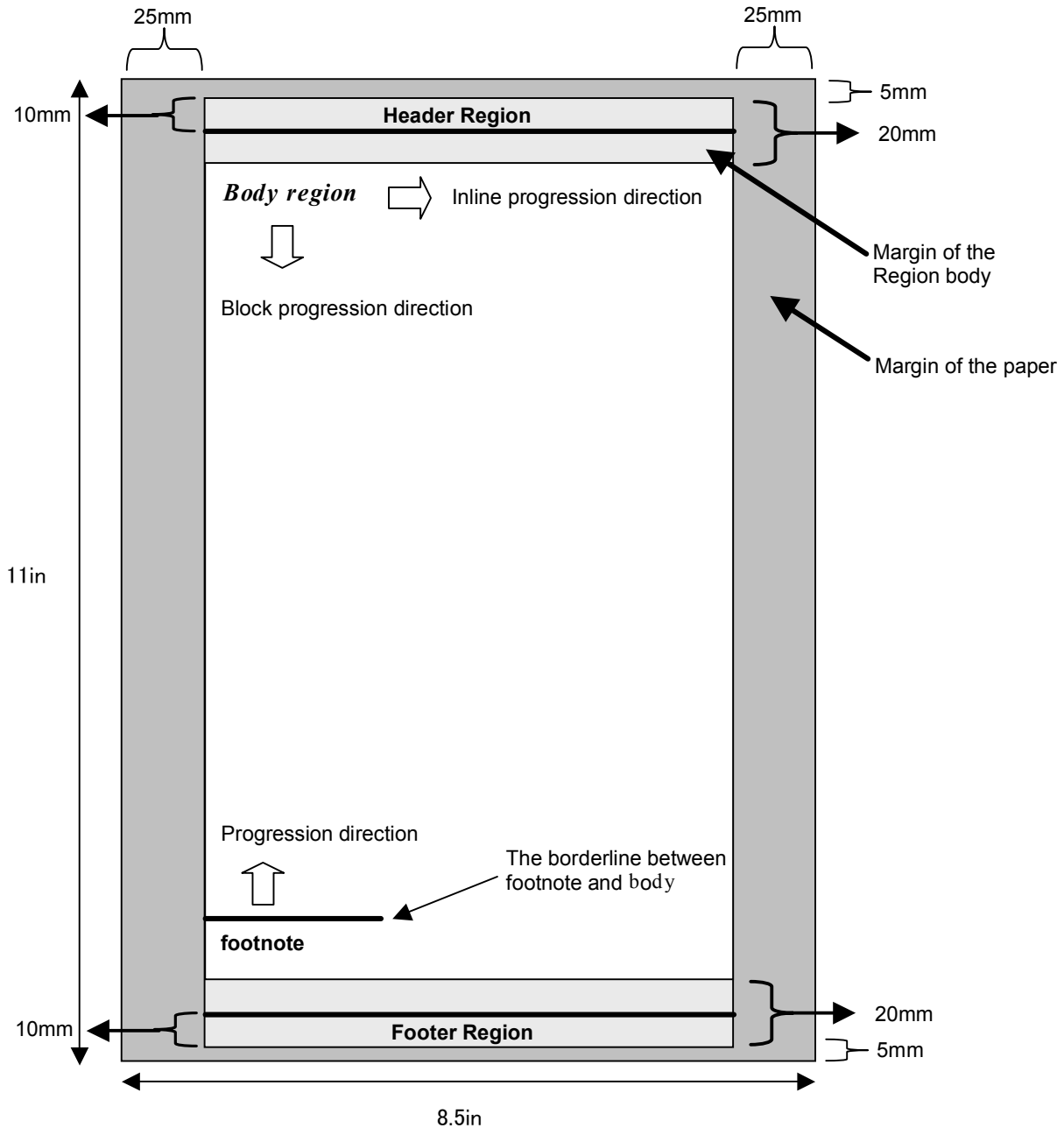
Item	Specification
Extent	10mm
Writing Mode	lr-tb
Content	Print the title Font size 9pt, It is center aligned, bottom aligned in the block progression derection.

【 Footer Region 】

Item	Specification
Extent	10mm
Writing Mode	lr-tb
Contents	Print the page numbers. Number format -n- Font size 9pt, It is center aligned, top aligned in the block progression direction.

【 Body Region 】

Item	Specification
Margin	Margin top：20mm, bottom：20mm, left：0mm, right：0mm
Content	Consists of head, table, list, paragraph, image.
Writing mode	lr-tb
Column	1
Default font size	10pt
Text align	justify
Other conditions	Place a borderline between a footnote region and a body region. The borderline is a solid, the length is one-third of the region body, left justified.



The Layout of the Region-body

The Page layout of the body region is fixed according to these instructions. The elements of the head and table should be also described in output specification. Although, we will explain these elements later individually . Beside the body part, a front cover and a content also should be specified, that have the same paper size, but defferent page layout from the body part.

Sample2fo.xsl Stylesheet

What is Sample2fo.xsl?

Sample2fo.xsl is a sample of a XSLT stylesheet that is made by revising Sdoc2fo.xsl for this SanpleDoc. Sdoc2fo.xsl is put on the Web by our company as a sample XSLT stylesheet for transforming PureSmartDoc into XSL-FO. Sample2fo.xsl achieves the output specification described in the last chapter, processes book covers, contents, headers, hooters, footnotes, images and so on.

Sample2fo.xsl Structure

Sample2fo.xsl consists of the following top level XSLT elements.

XSLT elements	Contents, Usage
xsl:param	Specify the paper size in the whole stylesheet as a parameter.
xsl:attribute-set	Specify one or more attributes as members of an attribute set, when you may apply the same group of attributes to many different elements.
xsl:template match="xxx"	Template for trasformation, template matchs each nodes of source XML documment. It is called by <xsl:apply-templates />
xsl:template name="yyy"	This is a subroutine which is explicitly invoked by <xsl:call-template name="yyy"/>

xsl:param, xsl:attribute-set is not necessarily required, but they have the following features:

- The stylesheet can be easy to see, easy to be handled by separating their roles. Properties of formatting objects are defined by xsl:attribute-set, while the transformation are defined by xsl:template.
- xsl:param provides a parameter at the time application calls the XSLT processor. It is possible to control a stylesheet processing externally depending upon the value of xsl:param.

An example of using xsl:param

```
<!-- Specify whether to make a table of contents or not -->
<xsl:param name="toc-make" select="false()"/>
<!-- Specify a paper size -->
<!-- Refer to $paper-width, $paper-height for the varue. -->
<xsl:param name="paper-width">8.5in</xsl:param>
<xsl:param name="paper-height">11in</xsl:param>
```

An example of using xsl:attribute-set

```
<!-- Specify the property of formatting object that match to the p element -->
 <!-- Refer to xsl:use-attribute-sets="p" -->
<xsl:attribute-set name="p">
  <xsl:attribute name="text-indent">1em</xsl:attribute>
  <xsl:attribute name="space-before">0.6em</xsl:attribute>
  <xsl:attribute name="space-after">0.6em</xsl:attribute>
  <xsl:attribute name="text-align">justify</xsl:attribute>
</xsl:attribute-set>
```

From now on, we will explain the stylesheet according to this Sample2fo.xsl.

Page Format Specification

Output Specification

- There are three page formats, front cover, table of contents and body.
- Page format for the body is defined in the print form specification.
- The page format for the front cover and the table of contents are basically the same as that of the body. But the page numbers and the document name are not included. Therefore they have no header, footer regions.

The XSLT stylesheet is shown below.

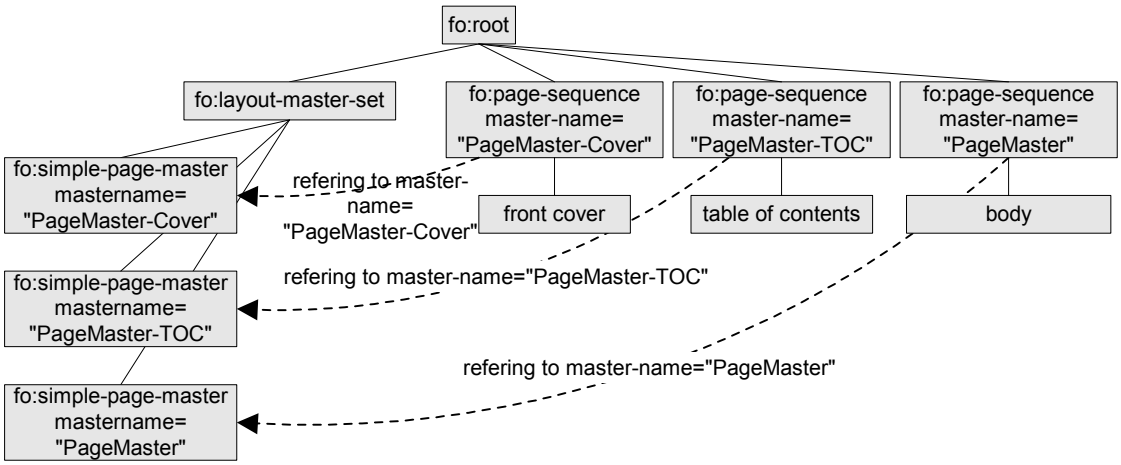
Definition of page layouts in the stylesheet

```
<fo:layout-master-set>
  <fo:simple-page-master margin="5mm 25mm 5mm 25mm" master-name="PageMaster">
    <xsl:attribute name="page-height">
      <xsl:value-of select="$paper-height"/>
    </xsl:attribute>
    <xsl:attribute name="page-width">
      <xsl:value-of select="$paper-width"/>
    </xsl:attribute>
    <fo:region-body   margin="20mm 00mm 20mm 00mm"/>
    <fo:region-before  border-after-style="solid" border-width="1pt" extent="10mm"
display-align="after"/>
    <fo:region-after  border-before-style="solid" border-width="1pt" extent="10mm"
display-align="before"/>
  </fo:simple-page-master>

  <fo:simple-page-master  margin="25mm 25mm 25mm 25mm" master-name="PageMaster-Cover">
    <xsl:attribute name="page-height">
      <xsl:value-of select="$paper-height"/>
    </xsl:attribute>
    <xsl:attribute name="page-width">
      <xsl:value-of select="$paper-width"/>
    </xsl:attribute>
    <fo:region-body  margin="0mm 0mm 0mm 0mm"/>
  </fo:simple-page-master>

  <fo:simple-page-master margin="25mm 25mm 25mm 25mm"  master-name="PageMaster-TOC">
    <xsl:attribute name="page-height">
      <xsl:value-of select="$paper-height"/>
    </xsl:attribute>
    <xsl:attribute name="page-width">
      <xsl:value-of select="$paper-width"/>
    </xsl:attribute>
    <fo:region-body  margin="0mm 0mm 0mm 0mm"/>
  </fo:simple-page-master>
</fo:layout-master-set>
```

Master name properties of fo:simple-page-master are set as follows:

master-name	Use	The template that refers this simple-page-master
PageMaster-Cover	For a front cover	<xsl:template match="doc/head">
PageMaster-TOC	Page format for table of contents	<xsl:template name="toc">
PageMaster	Page format for body	<xsl:template match="body">

The XSL-FO tree must be arranged as follows by using this page master.



Fo tree structure relating to page layout

Output Control of the Whole Stylesheet

Requirement for processing conditions

- Fo tree is generated in order of fo:page-sequence of the front cover, fo:page-sequene of the table of contents, fo:page-sequence of body.
- The front cover and the table of contents cannot be created by the stylesheet made in order of the XML source document, so you have to make the subroutine templates that create the front cover and the table of contents.
- These processings are controled by the templates that process the doc element, that is the root element.

The templates that process the doc elements created according to the requirement for processing conditions are shown below. And the order is fo:layout-master-set outputs, making a front cover, making a table of contens, processing the body text. In the process of writing documents, a front page and a table of contens are not everytime necessary. It is possible for the attributes of doc elements to control whether to make these or not.

The templates that process the doc elements

```
<xsl:param name="toc-make" select="false()"/>
<xsl:param name="cover-make" select="false()"/>

<xsl:template match="doc">
  <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
    <xsl:call-template name="xml-lang" />
    <fo:layout-master-set>
      <!-- Specify the page layout(fo:simple-page-master). This part is omitted-->
    </fo:layout-master-set>
    <!-- Make a front cover processed by the head elements. -->
    <xsl:if test="$cover-make or @cover!='false'">
      <xsl:apply-templates select="head" />
    </xsl:if>

    <!-- Call the template that creates a table of contents. -->
    <xsl:if test="$toc-make or @toc!='false'">
      <xsl:call-template name="toc" />
    </xsl:if>

    <!-- Process the text (descendants of the body elements) -->
    <xsl:apply-templates select="body" />
  </fo:root>
</xsl:template>
```

Creating a Front Cover

Output Specification

- Output contents of the title, the author, the date, those are children of the head element in the front page but not output the abstruct.
- The width of the block that contains the title is 130mm, the height is 20mm, and the block must be centered, using gray for the background color, dark gray for the border color. The title is placed 25mm down from the margin top, and make a 122mm height space between the title and the author to be written next. Use the font size 24pt, font style Arial. the text must be centered inside the block.
- The width of the block that contains the date is 160mm and the block must be centered, using no background color, no border. Use the font size 14pt, font style Times New Roman. Make a 5mm height space between the date and the author.
- The width of the block that contains the author is 160mm and the block must be centered, using no background color, no border. Use the font size 14pt, font style Times New Roman. When a image of logotype is specified to the author, put it preceding the author.

The front cover is created by the templates that process the head.

Format attributes of title, auther, date.

```
<!-- cover -->
<xsl:attribute-set name="cover.title" >
  <xsl:attribute name="space-before">25mm</xsl:attribute>
  <xsl:attribute name="space-before.conditionality">retain</xsl:attribute>
  <xsl:attribute name="space-after">122mm</xsl:attribute>
  <xsl:attribute name="font-size">24pt</xsl:attribute>
  <xsl:attribute name="font-family">"Arial"</xsl:attribute>
  <xsl:attribute name="text-align">center</xsl:attribute>
  <xsl:attribute name="text-align-last">center</xsl:attribute>
  <xsl:attribute name="start-indent">18mm</xsl:attribute>
  <xsl:attribute name="width">130mm</xsl:attribute>
  <xsl:attribute name="height">20mm</xsl:attribute>
  <xsl:attribute name="background-color">#EEEEEE</xsl:attribute>
  <xsl:attribute name="border-style">outset</xsl:attribute>
  <xsl:attribute name="border-color">#888888</xsl:attribute>
  <xsl:attribute name="padding-top">5pt</xsl:attribute>
  <xsl:attribute name="padding-bottom">5pt</xsl:attribute>
</xsl:attribute-set>

<xsl:attribute-set name="cover.date" >
  <xsl:attribute name="space-after">5mm</xsl:attribute>
  <xsl:attribute name="font-size">14pt</xsl:attribute>
  <xsl:attribute name="font-family">"Times New Roman"</xsl:attribute>
  <xsl:attribute name="text-align">center</xsl:attribute>
  <xsl:attribute name="text-align-last">center</xsl:attribute>
  <xsl:attribute name="width">160mm</xsl:attribute>
</xsl:attribute-set>

<xsl:attribute-set name="cover.author" >
  <xsl:attribute name="font-size">14pt</xsl:attribute>
  <xsl:attribute name="font-family">"Times New Roman"</xsl:attribute>
  <xsl:attribute name="text-align">center</xsl:attribute>
```

```
  <xsl:attribute name="text-align-last">center</xsl:attribute>
  <xsl:attribute name="width">160mm</xsl:attribute>
</xsl:attribute-set>
```

The templates that transform the head elements

```
<xsl:template match="doc/head">
  <fo:page-sequence master-name="PageMaster-Cover">
    <fo:flow flow-name="xsl-region-body" >
     <fo:block-container  xsl:use-attribute-sets="cover.title">
       <xsl:apply-templates select="/doc/head/title"/>
     </fo:block-container>
      <fo:block-container   xsl:use-attribute-sets="cover.date">
        <xsl:apply-templates select="/doc/head/date"/>
      </fo:block-container>
      <fo:block-container   xsl:use-attribute-sets="cover.author">
        <xsl:apply-templates select="/doc/head/author"/>
      </fo:block-container>
    </fo:flow>
  </fo:page-sequence>
</xsl:template>

<xsl:template match="doc/head/title">
  <fo:block>
    <xsl:apply-templates />
  </fo:block>
</xsl:template>

<xsl:template match="doc/head/date">
  <fo:block>
    <xsl:apply-templates />
  </fo:block>
</xsl:template>

<xsl:template match="doc/head/author">
  <fo:block>
    <xsl:if test="@logo" >
      <xsl:call-template name="author.logo.img"/>
    </xsl:if>
    <xsl:apply-templates />
  </fo:block>
</xsl:template>

<xsl:template name="author.logo.img">
  <xsl:choose>
    <xsl:when test="@pos='side'">
      <fo:inline space-end="1em">
        <fo:external-graphic src="{@logo}">
          <xsl:if test="@width and @height">
            <xsl:attribute name="content-width" >
              <xsl:value-of select="@width" />
            </xsl:attribute>
            <xsl:attribute name="content-height">
              <xsl:value-of select="@height" />
            </xsl:attribute>
          </xsl:if>
        </fo:external-graphic>
      </fo:inline>
```

```
    </xsl:when>
    <xsl:otherwise>
      <fo:block space-after="1em">
        <fo:external-graphic src="{@logo}">
          <xsl:if test="@width and @height">
            <xsl:attribute name="content-width" >
              <xsl:value-of select="@width" />
            </xsl:attribute>
            <xsl:attribute name="content-height">
              <xsl:value-of select="@height" />
            </xsl:attribute>
          </xsl:if>
        </fo:external-graphic>
      </fo:block>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

The template structure is very simple. Only apply the templates first to the title, then the date, and then the author.

In order to layout the title, we use fo:block-container. The width and the height of fo:block-container are specified as 130mm and 20mm. The width of the text region is 215.9mm(8.5in) - 25mm - 25mm = 165.9mm. It is much better if there is a function to center fo:block-container itself, but there is not. Subtract 130mm (the width of fo:block-container) from 165.9mm. Devide 35.9mm (the result)into two halves and then specify the start-indend = 18mm.

The title block is the first block in the region-body, so it is replaced just under the margin top by optimization even if space-befor="25mm" is specified. You can make a space forcibly by specifying space-before. conditionality="retain".

If the logo attribute is specified to the author, it is rendered as the image. This is processed by the author.logo.img template. The pos attribute specify to put the image on the left or top of the author. See the title of this report for an example.

Creating a Table of Contents

Output Specification

- The title of parts, chapters, sections, and subsections in the source XML document are output for a table of contents.
- Table of contents is positioned next to the front cover by feeding a page. The title is "Table of Contents". The background color is gray.
- The contents of each line consist of the each title in the part, chapter, section, subsection, and leaders (rows of dots), a page number.
- Space before, left indent, font size, font weight are specified in each line according to the nest level of each part, chapter, section, subsection.

Templates for Creating Contents

The table of contents is created by the toc template. The toc template is called from the templates that process the root elements doc, by using <xsl:call-template name="toc">

Toc template

```
<xsl:template name="toc">
  <!-- generate fo:page-sequence-->
    <fo:page-sequence master-name="PageMaster-TOC">
    <!-- generate flow applied to region-body -->
    <fo:flow flow-name="xsl-region-body" >
      <!--generate a block of table of contents-->
      <fo:block xsl:use-attribute-sets="div.toc">
        <!--generate the title｢Table of Contents｣ -->
        <fo:block xsl:use-attribute-sets="h2">Table of Contents</fo:block>
        <!-- select the elements of part, chapter, section, subsection,
                 subsubsection from the whole XML documents-->
        <xsl:for-each select="//part |
                              //chapter |
                              //section |
                              //subsection |
                              //subsubsection">
          <!-- apply template for each element to generate each line of the
contents.-->
          <xsl:call-template name="toc.line"/>
        </xsl:for-each>
      </fo:block>
    </fo:flow>
  </fo:page-sequence>
</xsl:template>
```

The toc template processes in the following order.

1. Create a new page-sequence. This page-sequence refer to fo:simple-page-master described in master-name="PageMaster-TOC" for a new page layout. Because the new page-sequence is created, the page is broken when it is printed.

2. Next, generate fo:flow object in the region-body. Create a block that contains a whole table of contents by applying attribute-set that is a name of div.toc. This attribte-set specifies the background color gray. Then, create a title "Table of Contents".
3. xsl:for-each select="..." is used to create a set of nodes consisting of the parts, characters, sections, subsections, subsubsection of whole document. Then, each node is sent to toc.line template that processes a line of the contents. In this processing, xsl:sort is not specified, the nodes in the set are created in appearing order.

This template is called from the template that processes the doc elements. So, "current node" is doc element node. xsl:for-each change this current node into each node group specified by the select attribute. Therefore, current node is one of the five elements, part, chapter, section, subsection, and subsubsection in the toc.line template. When xsl:for-each finished processing, the current node returns to the previous doc element node.

Templates that Create Lines of TOC

The toc.line template creates a line of contents.

The toc.line template creates each line of contents.

```
<!-- global parameter and variable used when creating the table of contents. -->
<xsl:param name="toc-level-default" select="3"/>
<!-- The template that creates the table of contents -->
<xsl:variable name="toc-level-max">
  <xsl:choose>
    <xsl:when test="not (doc/@toclevel)">
      <xsl:value-of select="$toc-level-default"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:value-of select="number(doc/@toclevel)"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:variable>

<xsl:template name="toc.line">
  <!-- Count the nest level of current node,
    set the value to "level" local variable. -->
  <xsl:variable name="level" select="count(ancestor-or-self::part |
                                           ancestor-or-self::chapter |
                                           ancestor-or-self::section |
                                           ancestor-or-self::subsection |
                                           ancestor-or-self::subsubsection )" />
 <!-- Test if the nest level can be a target. -->
  <xsl:if test="$level &lt;= $toc-level-max">
    <!-- Create fo:block for each line of toc.  -->
    <fo:block  text-align-last="justify">
      <!-- Widen the margin left in proportion to a nest level.-->
      <xsl:attribute name="margin-left">
        <xsl:value-of select="$level - 1"/>
        <xsl:text>em</xsl:text>
      </xsl:attribute>

      <!-- space-before becomes larger in proportion
           that the nest level becomes upper.-->
      <xsl:attribute name="space-before">
          <xsl:choose>
          <xsl:when test="$level=1">5pt</xsl:when>
          <xsl:when test="$level=2">3pt</xsl:when>
```

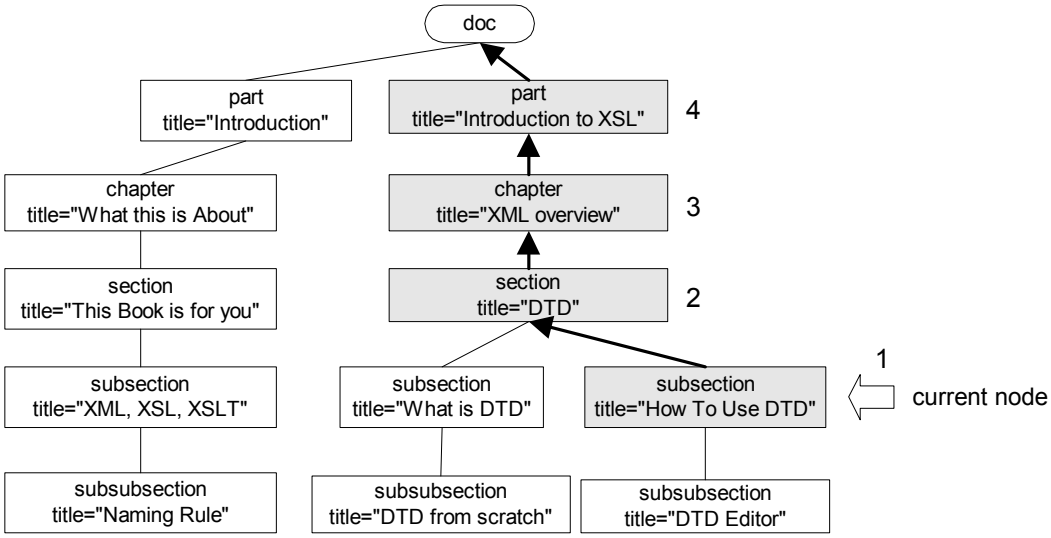
```
          <xsl:when test="$level=3">1pt</xsl:when>
          <xsl:otherwise>1pt</xsl:otherwise>
        </xsl:choose>
      </xsl:attribute>
      <!-- font-size is processed in the same way-->
      <xsl:attribute name="font-size">
        <xsl:choose>
          <xsl:when test="$level=1">1em</xsl:when>
          <xsl:otherwise>0.9em</xsl:otherwise>
        </xsl:choose>
      </xsl:attribute>
      <!-- font-weight is also processed in the same way -->
      <xsl:attribute name="font-weight">
        <xsl:value-of select="800 - $level * 100"/>
      </xsl:attribute>
      <!-- Below is the data of the table of contents -->
      <xsl:value-of select="title" />
      <fo:leader leader-pattern="dots"/>
     <!-- Output fo:page-number-citation. When the page is printed,
          it is replaced by the page number.-->
      <fo:page-number-citation ref-id="{generate-id()}"/>
    </fo:block>
  </xsl:if>
</xsl:template>
```

The toc.line template processes in the following order.

1. Count the depth of the nest of the current node (nest level) from the root node . Set the value to a level variable.
2. If a nest level is on or below the level of "toc-level-max", it is processed. If not, it is not processed. The level of "toc-level-max" is specified by the toclevel attribute of the doc element. If the level is not specified, the varue is 3.
3. Create fo:block for each line in the table of the contents.
4. According to the depth from the root node, determine the the property value of indent, font size, font weight.
5. Output contents of title elements, leaders, page numbers.

Count the Nest Level

The nest level can be counted by setting the local variable called level as follows: count(ancestor-or-self::part| ancestor-or-self::chapter|ancestor-or-self::section|ancestor-or-self::subsection|ancestor-or-self::subsubsection) In other words, count itself or the ancestor nodes. This chart is shown below:



※count(ancestor-or-self::part|ancestor-or-self::chapter|ancestor-or-self::section|ancestor-or-self::subsection) returns the number of the descendand nodes, including itself, of part, chapter, section, subsection, subsubsection.
For example in case that current node is title="How To Use DTD", count(...) function returns the value 4.

Count the nest level from the root element

Set properties according to the nest level.

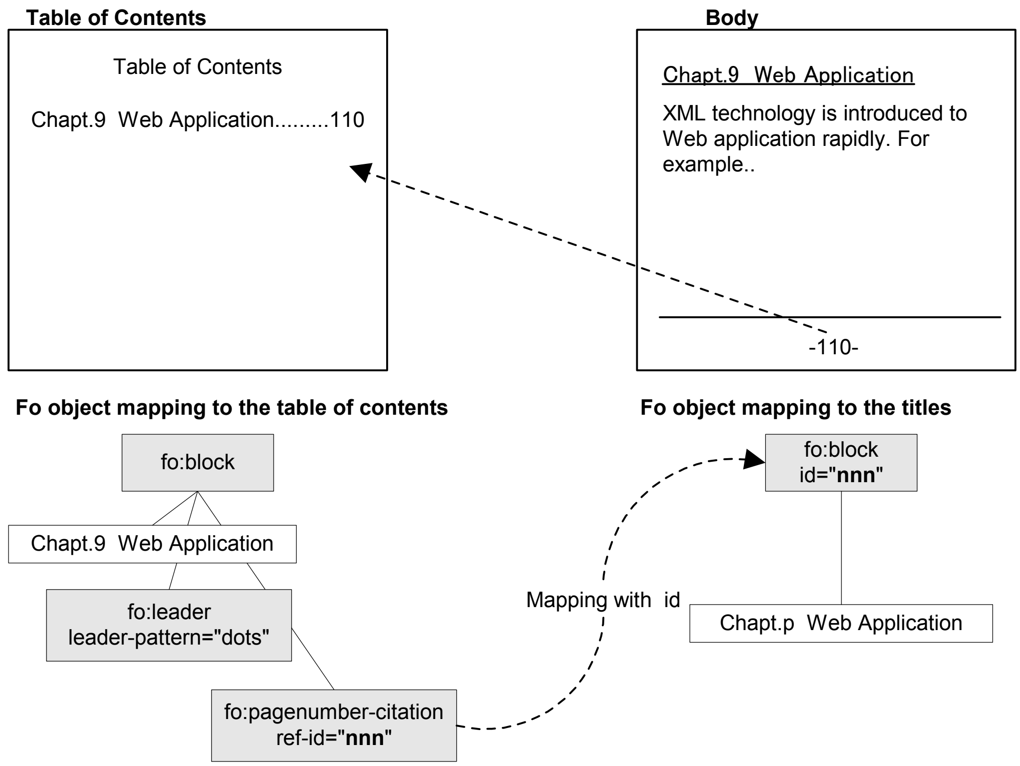
Set fo:block properties according to the nest level of the current node. Note that in this case, the properties are not set according to the element such as part, chapter, section, subsection. By setting properties according to the nest levels, the table of contents can be generated without depending on the elements used, but using the same format. Next table shows the properties set in the stylesheet.

Properiy	Nest level				
	1	2	3	4	5
margin-left	0em	1em	2em	3em	4em
space-before	5pt	3pt	1pt	1pt	1pt
font-size	1em	0.9em	0.9em	0.9em	0.9em
font-weight	700	600	500	400	300

Create page numbers

You have to create page numbers that show the page where each part,chapter, section, subsection appear. **Page numbers cannot be fixed until the formatter pagenates the XSL-FO instance.**

For the purpose, XSL provides the function fo:page-number-citation. The formatter replaces fo:page-number-citation with the page number in the end of the processing. The ref-id property specifies which page number is replaced. The formatter finds formatting object that has the same value in the id property as that specified by ref-id. Then, the page number that the formatting object belongs to are given. Therefore fo:block generated from the part, chapter, section, subsection elements must have id properties. This mechanism is shown below.



Mechanism of fo:page-number-citation

In the template, the generate-id() function is used as the value of the ref-id property. XSLT processor generates the unique characters using generate-id() to distinguish current node.

fo:leader

Use fo:leader between the title of the contents and the page numbers. fo:leader is a special object for generating inline area. In the example, leader-pattern="dots" is specified. It plays the role to fill the space between the title and the page number. It is important that text-align-last="justify" in fo:block specifies to justify the entire line. So, the titles are left-justified and the page numbers are right-justified, then the leader pattern fill the space between them.

fo: leader property can specify various patterns as shown below. fo:leader properties are written in the left side.

leader-pattern="dots"... 99
leader-pattern="use-content" (content="*")*** 99
leader-pattern="rule" rule-style="dotted"... 99
leader-pattern="rule" rule-style="dashed"- 99
leader-pattern="rule" rule-style="solid"____ 99
leader-pattern="rule" rule-style="double"____ 99
leader-pattern="rule" rule-style="groove"____ 99
leader-pattern="rule" rule-style="ridge"____ 99

Example of the generated contents

Shown below is an example of toc created taking the steps described.

Generated table of contents

```
<fo:block text-align-last="justify" margin-left="0em" space-before="5pt" font-
size="1em" font-weight="700">
  Preface
  <fo:leader leader-pattern="dots"/>
  <fo:page-number-citation ref-id="IDA4AIOB"/>
</fo:block>
```

See the table of contents in this report for an output example.

Processing the Body

Output specification

- Process all the descendants or self of the body elements in the source XML document.
- The page layout of the body is as described in the Printing Form Specification.

The Template Processing the Body

The body in the XML source document is contained to the decendants or self of the body element. Shown below are the templates that process the body element.

The template processing the body element

```
<xsl:template match="body">
  <!-- Create a new fo:page-sequence -->
  <fo:page-sequence master-name="PageMaster">
    <!-- Place the document title in the header region. -->
    <fo:static-content flow-name="xsl-region-before">
      <fo:block text-align="center" font-size="9pt">
        <xsl:if test="/doc/head/title">
          <xsl:value-of select="/doc/head/title"/>
        </xsl:if>
      </fo:block>
    </fo:static-content>

    <!-- Place the page numbers in the footer region. -->
    <fo:static-content flow-name="xsl-region-after">
      <fo:block text-align="center" font-size="9pt">
    - <fo:page-number/> -
      </fo:block>
    </fo:static-content>

    <!-- Place a borderline between body and footnote. -->
    <fo:static-content flow-name="xsl-footnote-separator">
      <fo:block>
        <fo:leader leader-pattern="rule" rule-thickness="0.5pt" leader-length="33%"/>
      </fo:block>
    </fo:static-content>

    <!-- Create the flow objects of the body region. -->
    <fo:flow flow-name="xsl-region-body" >
      <fo:block>
        <!-- Process the descendants or self of the body element. -->
        <xsl:apply-templates />
      </fo:block>
    </fo:flow>
  </fo:page-sequence>
</xsl:template>
```

This template pocesses as shown below.

1. Generate a fo:page-sequence based on the new "PageMaster", the page format is changed right after the table of contents.

2. Set the header/footer regions based on the page format. Place the title of the document in the header, page numbers in the footer.
3. Create a border region between the body and the footnote by using leaders.
4. Create flow objects in the body region.
5. xsl:apply-templates processes the descendants or self of the body element.

In order to print the page numbers, use fo:page-number object. fo:page-number creates the special inline area, the formatter replaces fo:page-number by the page number in the formatting process.

Use fo:leader object to create a borderline between the body and the footnote. Use solid line. The width of the line is one - third of the body region.

Creating Heads

Output specification

- Generate heads from the title of parts, chapters, sections, subsections and subsubsections.
- The style of the head is not mapped for each element of part, chapter, section, subsection and subsubsection but it is mapped according to the nest level.
- The head of the top level breaks page before the block.
- Make it possible to set an image in the head.

Style Conditions of the Head

Generally the style of the head is made according to the part, chapter, section, subsection and subsubsection elements, but in this case it is made according to the nest level. The conditions to be set are shown below.

Nest level	attribute-set	Style conditions
1	h1	font: size 24pt, Arial, Bold space-after : 14pt, keep-with-next.within-page : always botom border : solid 2pt, break condition : break-before="page"
2	h2	Font : size 16pt, Arial, bold space-before : 19pt, space-after : 5pt, keep-with-next. within-page : always
3	h3	font : size 13pt, Arial, bold space-before : 14pt, space-after : 5pt, keep-with-next. within-page : always
4	h4	font : size 12pt, Arial, bold space-before : 5pt, space-after: 5pt, keep-with-next. within-page : always
5	h5	font : size 10pt, Arial, bold space-before : 3pt, space-after : 3pt, keep-with-next. within-page : always

These conditions of style are defined in the following stylesheet.

Style definition of the heads

```
<!-- titles -->
<xsl:attribute-set name="h1" >
  <xsl:attribute name="font-size">24pt</xsl:attribute>
  <xsl:attribute name="font-family">"Arial"</xsl:attribute>
  <xsl:attribute name="font-weight">bold</xsl:attribute>
  <xsl:attribute name="space-after">14pt</xsl:attribute>
  <xsl:attribute name="break-before">page</xsl:attribute>
  <xsl:attribute name="keep-with-next.within-page">always</xsl:attribute>
  <xsl:attribute name="border-after-style">solid</xsl:attribute>
  <xsl:attribute name="border-after-width">2pt</xsl:attribute>
</xsl:attribute-set>

<xsl:attribute-set name="h2" >
  <xsl:attribute name="font-size">16pt</xsl:attribute>
  <xsl:attribute name="font-family">"Arial"</xsl:attribute>
  <xsl:attribute name="font-weight">bold</xsl:attribute>
  <xsl:attribute name="space-before">19pt</xsl:attribute>
```

```
  <xsl:attribute name="space-after">5pt</xsl:attribute>
  <xsl:attribute name="keep-with-next.within-page">always</xsl:attribute>
</xsl:attribute-set>

<xsl:attribute-set name="h3" >
  <xsl:attribute name="font-size">13pt</xsl:attribute>
  <xsl:attribute name="font-family">"Arial"</xsl:attribute>
  <xsl:attribute name="font-weight">bold</xsl:attribute>
  <xsl:attribute name="space-before">14pt</xsl:attribute>
  <xsl:attribute name="space-after">5pt</xsl:attribute>
  <xsl:attribute name="keep-with-next.within-page">always</xsl:attribute>
</xsl:attribute-set>

<xsl:attribute-set name="h4" >
  <xsl:attribute name="font-size">12pt</xsl:attribute>
  <xsl:attribute name="font-family">"Arial"</xsl:attribute>
  <xsl:attribute name="font-weight">bold</xsl:attribute>
  <xsl:attribute name="space-before">5pt</xsl:attribute>
  <xsl:attribute name="space-after">5pt</xsl:attribute>
  <xsl:attribute name="keep-with-next.within-page">always</xsl:attribute>
</xsl:attribute-set>

<xsl:attribute-set name="h5" >
  <xsl:attribute name="font-size">10pt</xsl:attribute>
  <xsl:attribute name="font-family">"Arial"</xsl:attribute>
  <xsl:attribute name="font-weight">bold</xsl:attribute>
  <xsl:attribute name="space-before">3pt</xsl:attribute>
  <xsl:attribute name="space-after">3pt</xsl:attribute>
  <xsl:attribute name="keep-with-next.within-page">always</xsl:attribute>
</xsl:attribute-set>
```

keep-with-next.within-page="always" is specified to heads in order to avoid breaking pages before the next block. It is not specified in h1, instead, break-before="page" is specified, so the page break is inserted before h1 block, a block is kept at the top of the following page and is kept with the next one.

Head Processing Templates

The templates that process the heads. Heads are processed in one template intensively because the styles are selected according to the nest level.

The templates that process heads

```
<xsl:template match="part |
                     chapter |
                     section |
                     subsection |
                     subsubsection">
  <xsl:call-template name="title.out"/>
  <xsl:apply-templates />
</xsl:template>

<xsl:template match="part/title |
                     chapter/title |
                     section/title |
                     subsection/title |
                     subsubsection/title">
</xsl:template>
```

```
<xsl:template name="title.out">
  <xsl:variable name="level" select="count(ancestor-or-self::part |
                                           ancestor-or-self::chapter |
                                           ancestor-or-self::section |
                                           ancestor-or-self::subsection |
                                           ancestor-or-self::subsubsection )" />
  <xsl:choose>
    <xsl:when test="$level=1">
      <fo:block xsl:use-attribute-sets="h1" id="{generate-id()}">
        <xsl:call-template name="title.out.sub"/>
        <xsl:value-of select="title"/>
      </fo:block>
    </xsl:when>
    <xsl:when test="$level=2">
      <fo:block xsl:use-attribute-sets="h2" id="{generate-id()}">
        <xsl:call-template name="title.out.sub"/>
        <xsl:value-of select="title"/>
      </fo:block>
    </xsl:when>
    <xsl:when test="$level=3">
      <fo:block xsl:use-attribute-sets="h3" id="{generate-id()}">
        <xsl:call-template name="title.out.sub"/>
        <xsl:value-of select="title"/>
      </fo:block>
    </xsl:when>
    <xsl:when test="$level=4">
      <fo:block xsl:use-attribute-sets="h4" id="{generate-id()}">
        <xsl:call-template name="title.out.sub"/>
        <xsl:value-of select="title"/>
      </fo:block>
    </xsl:when>
    <xsl:when test="$level=5">
      <fo:block xsl:use-attribute-sets="h5" id="{generate-id()}">
        <xsl:call-template name="title.out.sub"/>
        <xsl:value-of select="title"/>
      </fo:block>
    </xsl:when>
    <xsl:otherwise>
      <fo:block xsl:use-attribute-sets="h5" id="{generate-id()}">
        <xsl:call-template name="title.out.sub"/>
        <xsl:value-of select="title"/>
      </fo:block>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template name="title.out.sub">
  <xsl:if test="@logo">
    <fo:inline space-end="5pt">
      <fo:external-graphic src="{@logo}">
        <xsl:if test="@width and @height">
          <xsl:attribute name="content-width" >
            <xsl:value-of select="@width" />
          </xsl:attribute>
          <xsl:attribute name="content-height">
            <xsl:value-of select="@height" />
          </xsl:attribute>
        </xsl:if>
      </fo:external-graphic>
```

```
    </fo:inline>
  </xsl:if>
</xsl:template>
```

The stylesheet processing heads consists of four templates. Actual title of the head is created form the title.out template. The title.out template processes in the following order:

1. Count the nest level of the element under processing and store the value to a level local variable.
2. Select style of the title from h1 to h5 according to the variable, apply it to fo:block of the title.
3. In the same way, create id property by generate-id() function and apply it to fo:block. (5)
4. Call the title.out.sub template that processes images.
5. Output the text to the title as specified.

The second template that processes part/title - subsubsection/title become empty (nothing output). It is because the template has already processed by the title.out template and not to output the text of title when the first xsl:apply-templates invoke it again.

A generated example of the titles

This is a geneated example of title by taking previous steps. Note that the result of generate-id() funcion is stored as the value of id.

Created titles

```
<fo:block font-size="24pt"
          font-family="&quot;Arial&quot;"
          font-weight="bold"
          space-after="14pt"
          break-before="page"
          keep-with-next.within-page="always"
          border-after-style="solid"
          border-after-width="2pt"
          id="IDA4AIOB">
  <fo:inline space-end="5pt">
    <fo:external-graphic src="XSLFormatter.bmp"/>
  </fo:inline>
  Preface
</fo:block>
```

(5). Refer to the fo:page-number-citation of the line of the contents

Processing Inline Elements

Output specification

- b(bold), i(italic), em(emphasis), code(inline program code) are transformed into character properties.
- The a(anchor) element only output the referenced text first, output the contents of a target end of a hypertext link of the href attribute.
- A note(note) is transformed into a footnote. A footnote citation is usually (n). A sequence of numbers is applied to (n).
- The br(break) element breaks lines.
- The span element (general inline element) only creates fo:inline.

The Templates that Process b, i, em, code Elements.

It is very easy to transform b(bold), i(italic), em(emphasis), code(inline program code) into formatting objects. The templates create fo:inline template and set attributes to be applied. Bold is set as font-weight="bold", italic is font-style="italic", em also maps to bold. The way is different from the b but it becomes the same result. Code sets monospace to font-family porperty.

The templates that process b, i, em, code

```
<xsl:template match="b">
  <fo:inline font-weight="bold">
    <xsl:apply-templates />
  </fo:inline>
</xsl:template>

<xsl:template match="i">
  <fo:inline font-style="italic">
    <xsl:apply-templates />
  </fo:inline>
</xsl:template>

<xsl:template match="em">
  <fo:inline xsl:use-attribute-sets="em">
    <xsl:apply-templates />
  </fo:inline>
</xsl:template>

<xsl:template match="code">
  <fo:inline font-family="monospace">
    <xsl:apply-templates />
  </fo:inline>
</xsl:template>
```

As fo:inline is applied, the line is not considered to break at the last of text. The example is shown below.

The inline element i (italic) becomes [*Italic*].

The inline element (bold) becomes [**Bold typeface**].

The inline element em (emphasis) also becomes [**Bold typeface**].

The inline element code (inline program code) becomes [`Monospace font (apply monospace font)`].

The Anchor Element (a)

The anchor element (a) contains a Hypertext Reference attribute (href) which contains a URL. It is a problem how to treat a URL. In this case put a URL in parentheses, output it after the text specified by the anchor element. But if the both contents are the same, the URL is not output.

The templates that process the (a) element

```
<xsl:template match="a">
  <xsl:variable name="anchor-texts">
    <xsl:value-of select="." />
  </xsl:variable>
  <xsl:apply-templates />
  <xsl:if test="@href!=$anchor-texts">
    <fo:inline >
      <xsl:text>(</xsl:text>
      <xsl:value-of select="@href" />
      <xsl:text>)</xsl:text>
    </fo:inline>
  </xsl:if>
</xsl:template>
```

The example using this stylesheet is as follows:

'This example is shown on the Web at the site of W3C'.
is shown as:
'This example is shown on the Web at the site of W3C(http://www.w3.org/).'

'This example is shown on the Web site at http://www.w3.org/.'
is shown as:
'This example is shown on the Web site at http://www.w3.org/. '

'This example is shown on the Web site at http://www.w3.org/.'
is shown as:
'This example is shown on the Web site at http:// ***www.w3.org***/.'

The note Element

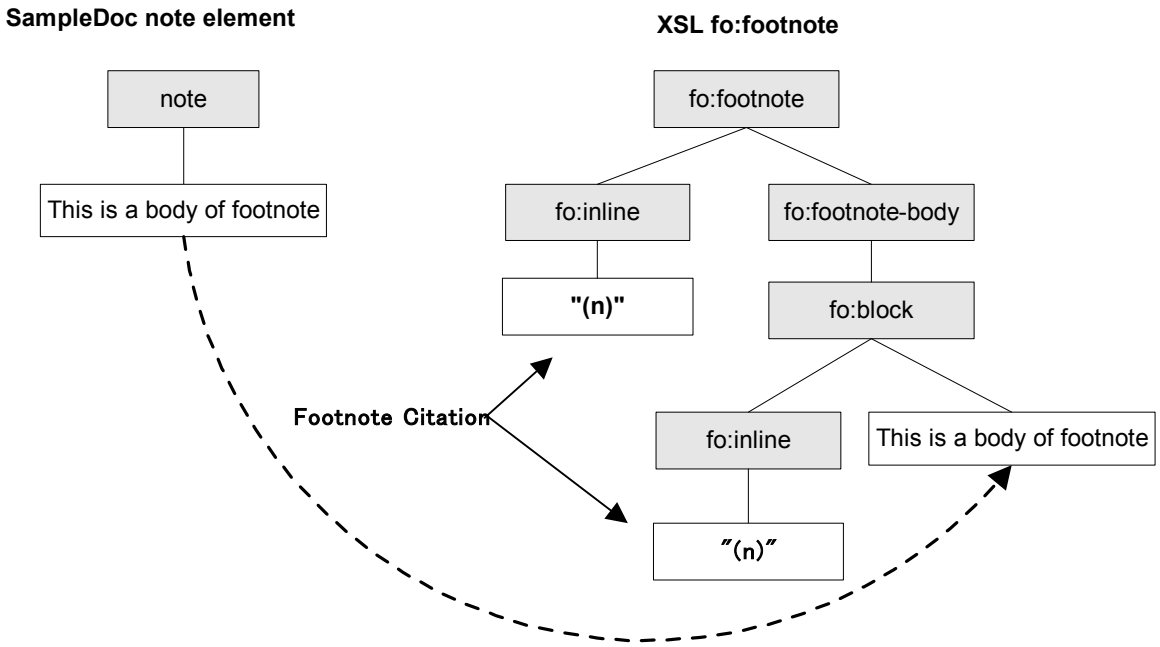
The note element is transformed into XSL fo:footnote formatting object.

The template that processes note element

```
<xsl:template match="note">
  <fo:footnote>
    <fo:inline baseline-shift="super" font-size="75%">
      <xsl:number level="any" count="//note" format="(1)" />
    </fo:inline>
```

```
    <fo:footnote-body>
      <fo:block xsl:use-attribute-sets="note">
        <fo:inline baseline-shift="super" font-size="75%">
          <xsl:number level="any" count="//note"  format="(1)"/>
        </fo:inline>
          <xsl:apply-templates />
      </fo:block>
    </fo:footnote-body>
  </fo:footnote>
</xsl:template>
```

fo:footnote object in the XSL express a footnote and the content model is (fo:inline, fo:footnote-body). The first child fo:inline expresses a footnote citation placed in the text. The next fo:footnote-body is the footnote text and it consists of the block objects like fo:block. Shown below is a typical fo:footnote object.



The typical example of a note element and a fo:footnote

Generaly, the same content as the footnote citation in the text is placed before the footnote text. and a footnote citation is usually a sequence of numbers. These characters must be generated in the stylesheet. Formatting object does not have the function to achieve these. The following shows the the stylesheet processing.

1. Output fo:footnote.
2. Generate fo:inline that includes a footnote citation by using xsl:number.
3. A content of note element with the same footnote citation is put into a fo:block element, that is a child of the fo: footnote-body.

xsl:number is a XSLT processing instruction. <xsl:number level="any" count="//note" format="(1)" />searchs all the descendant of the note elements under the root element in appearing order, find out the same elemet as the current note element. Formats the appearing number as described in "(1)",

baseline-shift="super" shifts the baseline to the default position for superscripts. Below shows the example of footnote.

"This is an example of footnote. Place a footnote here. <note> This is a footnote text. It is placed below the page and seperated from the text part.</note>" is expressed as:
"This is an example of footnote. Place a footnote here. [6]"

The br Element

The br element is an empty element, so it is replaced by a empty fo:block element. Then the line bleaks.

The template that processes the br elements

```
<xsl:template match="br">
  <fo:block>
  </fo:block>
</xsl:template>
```

The following table shows the example of the process

"In the paragraph, put forcibly
 break a line because the paragraph is not finished, the attribute set before the line break can be used."
is expressed as:

"In the paragraph, put **forcibly**
break a line because the paragraph is not finished, the attribute set before the line break can be used."

The span Element

The span attribute (general inline element) is only transformed into fo:inline. It can be extended if any instructions are defined for the class attribute

The template that processes the span element

```
<xsl:template match="span">
  <fo:inline >
    <xsl:apply-templates />
  </fo:inline>
</xsl:template>
```

[6]This is a footnote text. It is placed below the page and seperated from the text part.

Processing Block Elements

The processsing of the block elements except tables and lists (ol,ul,dl) are explained in this chapter.

Output specification

- Transform p(paragraph) into a block element. Indent one character in the first line of the paragraph. Text align justify. Text align left at the last line. Spaces which size is font size x 0.6 are given before and after the paragraph. A paragraph is kept within a page.
- Figures are placed by feeding a line at the appearing point, by centering. Apply the size of the figure if the size is specified. If there is a title, place it after the figure.
- Transform program element that contains program codes into fo:block formatting object. Apply monospace, use line feed, space as they are preformatted. The background color is gray. Output the title in front of the program code.
- Transform div(general block element) into fo:block.

The p Element

The p element (paragraph) is frequently used. The following shows the template processing the p element.

The template processing the p element

```
<xsl:attribute-set name="p">
  <xsl:attribute name="text-indent">1em</xsl:attribute>
  <xsl:attribute name="space-before">0.6em</xsl:attribute>
  <xsl:attribute name="space-after">0.6em</xsl:attribute>
  <xsl:attribute name="text-align">justify</xsl:attribute>
  <xsl:attribute name="keep-together.within-page">always</xsl:attribute>
</xsl:attribute-set>

<xsl:template match="p">
  <fo:block xsl:use-attribute-sets="p">
        <xsl:apply-templates />
  </fo:block>
</xsl:template>
```

The templates are simple, and is only to transform into fo:block. The indent in the begining of the line is specified by text-indent, the line justification is specified by text-align. And specify keep-together to keep a paragraph within one page.

This is the case that one line has the width of 21em.

This is the sample of a paragragph. The paragraph is transformed into fo:block. Specify that one character is indented by text-indent="1em". Specify that the content is to be expanded to both edges by text-align="justify". But the last sentence of the last paragraph is left justified. It's because the default value of text-indent-last is text-align="justifiy", it is justified automatically.

In case that one line has the width of 25em.

This is the sample of paragragph. The paragraph is transformed into fo:block. Specify that one character is indented by text-indent="1em". Specify that the content is to be expanded to both edges by text-align="justify". But the last sentence of the last paragraph is left justified. It's because the default value of text-indent-last is text-align="justifiy", it is justified automatically.

The figure Element

Figure elements are transformed into fo:external-graphic. The following shows the templates that process figure elements.

Templates that process the figure elements

```
<xsl:attribute-set name="figure.title" >
  <xsl:attribute name="font-family">sans-serif</xsl:attribute>
  <xsl:attribute name="text-align">center</xsl:attribute>
  <xsl:attribute name="space-before">3pt</xsl:attribute>
  <xsl:attribute name="space-after">10pt</xsl:attribute>
  <xsl:attribute name="font-weight">bold</xsl:attribute>
  <xsl:attribute name="keep-with-previous.within-page">always</xsl:attribute>
</xsl:attribute-set>

<xsl:template match="figure">
  <fo:block text-align="center">
    <fo:external-graphic src="{@src}">
      <xsl:if test="@width and @height">
        <xsl:attribute name="content-width" >
          <xsl:value-of select="@width"/>
        </xsl:attribute>
        <xsl:attribute name="content-height" >
          <xsl:value-of select="@height"/>
        </xsl:attribute>
      </xsl:if>
    </fo:external-graphic>
  </fo:block>
  <fo:block xsl:use-attribute-sets="figure.title">
```

```
  <xsl:value-of select="title" />
  </fo:block>
</xsl:template>
```

fo:external-graphic is generated and path of the figure is specified by the src attribute of figure element. In case the size of the figure is specified by the width, height attributes, put them in content-width, content-height. Antenna House XSL Formatter can treat image file formats such as BMP, EMF, WMF, JPEG.

The program Element

The program elements are transformed into fo:block and desplay the fonts in monospace. The following shows the templates.

Templates that process the program elements

```
<xsl:attribute-set name="program">
  <xsl:attribute name="white-space">pre</xsl:attribute>
  <xsl:attribute name="wrap-option">wrap</xsl:attribute>
  <xsl:attribute name="background-color">gainsboro</xsl:attribute>
  <xsl:attribute name="font-family">monospace</xsl:attribute>
  <xsl:attribute name="font-size">9pt</xsl:attribute>
  <xsl:attribute name="padding">0.5em</xsl:attribute>
</xsl:attribute-set>

<xsl:attribute-set name="program.title" >
  <xsl:attribute name="font-family">sans-serif</xsl:attribute>
  <xsl:attribute name="text-align">center</xsl:attribute>
  <xsl:attribute name="space-before">3pt</xsl:attribute>
  <xsl:attribute name="font-weight">bold</xsl:attribute>
  <xsl:attribute name="keep-with-next.within-page">always</xsl:attribute>
  <xsl:attribute name="space-before">0.5em</xsl:attribute>
  <xsl:attribute name="space-after">0.5em</xsl:attribute>
</xsl:attribute-set>

<xsl:template match="program">
  <xsl:apply-templates select="title"/>
    <fo:block xsl:use-attribute-sets="program">
      <xsl:apply-templates select="text()"/>
  </fo:block>
</xsl:template>
```

The program element and the p element has the common point that both generate fo:block in the template. But the difference is that the program element process only text(). But there are no templates to process text nodes (match="program/text()"). So they are processed by the built-in template in XSLT in order to process only text node. The following are important properties that apply to fo:block.

- Specify monospace in the font-family.
- Specify white-space as pre. This has four meanings as follows.
  1. linefeed-treatment="preserve"：Line feed character (#xA) is preserved. Do not treat as space, or ignore.
  2. space-treatment="preserve"：Characters classified as white space in XML except for #xA are preserved. [7]
  3. white-space-collapse="false"： White spaces after processing by linefeed-treatment, space-treatment are not collapsed.

[7]White space in XML has space(#x20), tab(#x9), carriage return(#xD), linefeed(#xA). The treatment of #xA is specified by linefeed-treatment

4. wrap-option="no-wrap"：No line-wrapping will occur if the line overflows.

- Specify wrap-option as wrap. The line wraps if the line overflows. [8]

According to these specifications the text in the program element is formatted as pre-formatted text.

The div Element

The div element(general block element) simply transforms into fo:block with no properties. The templates are shown below.

Template that processes the div element.

```
<xsl:template match="div">
  <fo:block>
    <xsl:apply-templates />
  </fo:block>
</xsl:template>
```

The example of applying the div element is as follows. This sample stores formatting object directly to the div element and output it.

Example:

```
<xsl:attribute-set name="div.fo" >
  <xsl:attribute name="border">solid</xsl:attribute>
  <xsl:attribute name="border-width">thin</xsl:attribute>
  <xsl:attribute name="padding">1em</xsl:attribute>
</xsl:attribute-set>

<xsl:template match="div[@class = 'fo']">
  <fo:block xsl:use-attribute-sets="div.fo">
    <xsl:copy-of select="node()" />
  </fo:block>
</xsl:template>
```

The template specifies to copy all the descendants of the div element directly to the output by <xsl:copy-of select="node()" />. In order to use this function, specify fo namespace to the doc element.

```
<doc xmlns:fo="http://www.w3.org/1999/XSL/Format">
```

Below shows how to use.

```
<div class="fo"><fo:block>This is the <fo:inline font-weight="bold">example</fo:inline> of embedding<fo:inline font-size="1.5em" text-decoration="underline" font-style="italic" font-weight="bold">FO (Formatting Object).</fo:inline><fo:inline background-color="#DDDDDD">You can format the styles as you like,</fo:inline> free from the <fo:inline font-size="1em">s</fo:inline><fo:inline font-size="1.2em">t</fo:inline><fo:inline font-size="1.4em">y</fo:inline><fo:inline font-size="1.6em">l</fo:inline><fo:inline font-size="1.8em">e</fo:inline><fo:inline font-size="2.0em">s</fo:inline><fo:inline font-size="2.2em">h</fo:inline><fo:inline font-size="2.4em">e</fo:inline><fo:inline font-size="2.6em">e</fo:inline><fo:inline font-size="2.8em">t</fo:inline>limitation. But it is very<fo:inline font-size="3em" font-weight="bold" font-family="sans-serif">troublesome</fo:inline>to write Fo directly．</fo:block></div>
```

It is shown as follows:

[8]wrap-option="wrap" is a default value. But it changes to wrap-option="no-wrap" by specifying white-space="pre". This property overide wrap-option="no-wrap" by wrap-option ="wrap"

This is the **example** of embedding ***FO (Formatting Object).*** You can format the styles as you like, free from the stylesheet limitation. But it is very **troublesome** to write Fo directly.

Processing table Elements

Output specification

- The descendants and self of a table element in the XML source document create a table formatting object.
- The background color of the head in the table is gray to descriminate from the contents in the table. And the line is solid, the line width is 1 pt.
- Cells in the table store the cell data placing the paddings which size is 0.3 x fontsize on the left, 0.2 x fontsize on the right.
- Process attributes specifying other format of the table.
  1. layout attribute, width attribute,rowheight attribute of the table element.
  2. number attribute, width attribute of the col element
  3. height attribute of the tr element
  4. align attribute, valign attribute, colspan attribute, rowspan attribute defined for th, td elements.

Comparing the SampleDoc Table with XSL Table

In order to process the descendants of the table element and generate a table, it should be transformed into fo:table-and-caption formatting object. Let us compare the SampleDoc table with XSL table. First of all, the SampleDoc table is as follows:

element	mean	definition
table	entire table	(title?, col*, thead?, tfoot?, tbody) Specify whether the table layout is formatted automatically or it is fixed, in the layout attribute.The entire width is specified in the width attribute, the entire height of the table is specified in the row height attribute.
col	column attribute	EMPTY The column width is specified in the width attribute, the column number is specified in the number attribute.
thead	table header	(tr*)
tfoot	table footer	(tr*)
tbody	table body	(tr*)
tr	table row	(th | td)* The height of the row is specified in the height attribute.
th	table header cell	(a group of inline elements)* The number of the rows to be expanded accross is specified in the colspan attribute, the number of the rows to be expanded down is specified in the rowspan. The align,valign attributes allows horizontal, vertical alignment to be set.
td	table data cell	(a group of inline elements)* The number of the rows to be expanded accross is specified in the colspan attribute, the number of the rows to be expanded down is specified in the rowspan. The align, valign attributes allows horizontal, vertical alignment to be set.

While, table of XSL consists of the following objects.

Element	Meaning	Definition
fo:table-and-caption	The whole table and the caption	(table-caption?, table)
fo:table-caption	The caption of the table	(%block;)[9]
fo:table	The grid composed of cells except caption cells.	(fo:table-column*, fo:table-header?, fo:table-footer?, fo:table-body+) table-layout：Specify automatical table layout or fixed layout. table-omit-header-at-break：Specify whether to omit placing header or not when the page breaks. table-omit-footer-at-break：Specify whether to omit placing footer or not when the page breaks.
fo:table-column	Define the feature of the table column	EMPTY column-number：The number of columns [10] column-width：The width of the column number-columns-repeated：The number of columns specifying the repetition of the table-column. number-colums-spanned：The number of columns spanned by table-cells.[11]
fo:table-header	Table header	(fo:table-row+|fo:table-cell+)
fo:table-footer	Table footer	(fo:table-row+|fo:table-cell+)
fo:table-body	Table body	(fo:table-row+|fo:table-cell+)
fo:table-row	Table row	(fo:table-cell+)
fo:table-cell	Table cell	(%block;)+ number-columns-spanned：The number of columns spanned by table-cells. number-rows-spanned : The number of rows spanned.

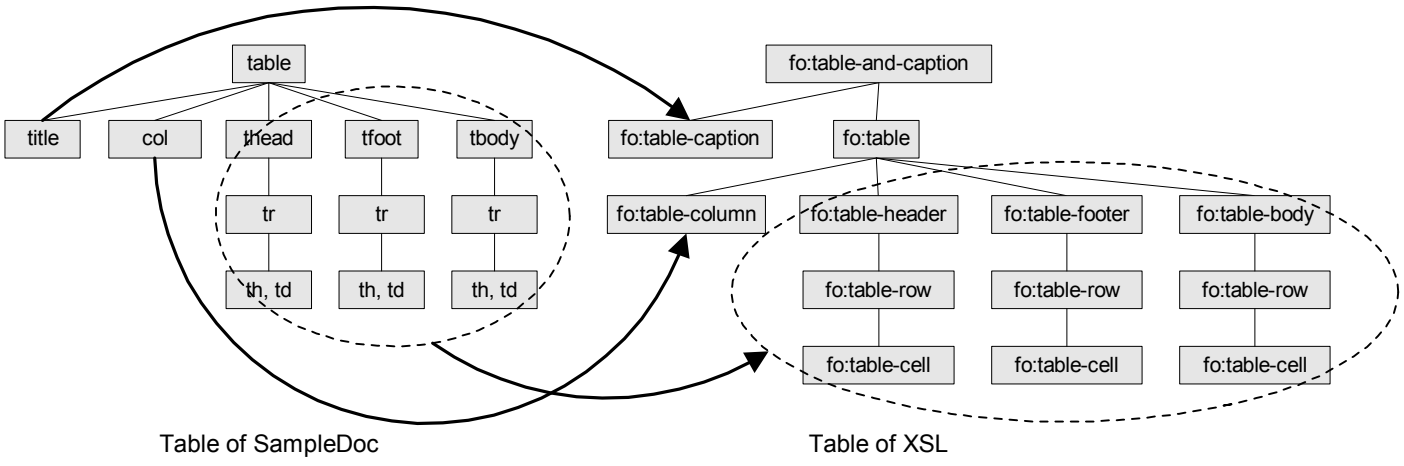


Table structure of SampleDoc, XSL-FO

[9] %block; is defined to contain block, block-container, table-and-caption, table, list-block in the XSL Specificaion. Generally, it is fo:block, fo:block-container.

[10] Use as a parmeter of from-table-column() function.

[11] Refer to from-table-column() function.

Comparing these two, they have almost the same structure. Transforming into tables is basically considered to exchange the structure to another.

Table Processing Templates

Templates that process table are shown below.

Definition of the table properties

```
<xsl:attribute-set name="table.data" >
  <xsl:attribute name="table-layout">fixed</xsl:attribute>
  <xsl:attribute name="space-before">10pt</xsl:attribute>
  <xsl:attribute name="space-after">10pt</xsl:attribute>
</xsl:attribute-set>

<xsl:attribute-set name="table.data.caption" >
  <xsl:attribute name="font-family">sans-serif</xsl:attribute>
  <xsl:attribute name="text-align">start</xsl:attribute>
  <xsl:attribute name="space-before">3pt</xsl:attribute>
  <xsl:attribute name="space-after">3pt</xsl:attribute>
  <xsl:attribute name="space-after.precedence">2</xsl:attribute>
  <xsl:attribute name="font-weight">bold</xsl:attribute>
  <xsl:attribute name="keep-with-next.within-page">always</xsl:attribute>
</xsl:attribute-set>

<xsl:attribute-set name="table.data.th" >
  <xsl:attribute name="background-color">#DDDDDD</xsl:attribute>
  <xsl:attribute name="border-style">solid</xsl:attribute>
  <xsl:attribute name="border-width">1pt</xsl:attribute>
  <xsl:attribute name="padding-start">0.3em</xsl:attribute>
  <xsl:attribute name="padding-end">0.2em</xsl:attribute>
</xsl:attribute-set>

<xsl:attribute-set name="table.data.td" >
  <xsl:attribute name="border-style">solid</xsl:attribute>
  <xsl:attribute name="border-width">1pt</xsl:attribute>
  <xsl:attribute name="padding-start">0.3em</xsl:attribute>
  <xsl:attribute name="padding-end">0.2em</xsl:attribute>
</xsl:attribute-set>
```

Templates that process the table

```
<xsl:template match="table">
  <fo:table-and-caption >
    <xsl:if test="title">
      <fo:table-caption xsl:use-attribute-sets="table.data.caption">
        <fo:block start-indent="0em">
          <xsl:apply-templates select="title" mode="make-title"/>
       </fo:block>
      </fo:table-caption>
    </xsl:if>
    <fo:table xsl:use-attribute-sets="table.data">
      <xsl:if test="@layout">
        <xsl:attribute name="table-layout">
          <xsl:value-of select="@layout"/>
        </xsl:attribute>
      </xsl:if>
```

```
      <xsl:if test="@width">
        <xsl:attribute name="inline-progression-dimension">
          <xsl:value-of select="@width"/>
        </xsl:attribute>
      </xsl:if>
      <xsl:apply-templates />
    </fo:table>
  </fo:table-and-caption>
</xsl:template>

<xsl:template match="table/tltle" mode="make-title">
  <xsl:apply-templates />
</xsl:template>

<xsl:template match="table/tltle">
</xsl:template><xsl:template match="col">
  <fo:table-column column-number="{@number}" column-width="{@width}" />
</xsl:template>

<xsl:template match="thead">
  <fo:table-header>
    <xsl:apply-templates />
  </fo:table-header>
</xsl:template>

<xsl:template match="tfoot">
  <fo:table-footer>
    <xsl:apply-templates />
  </fo:table-footer>
</xsl:template>

<xsl:template match="tbody">
  <fo:table-body>
    <xsl:apply-templates />
  </fo:table-body>
</xsl:template>

<xsl:template match="tr">
  <xsl:element name="fo:table-row">
    <xsl:choose>
      <xsl:when test="@height">
        <xsl:attribute name="block-progression-dimension">
          <xsl:value-of select="@height"/>
        </xsl:attribute>
      </xsl:when>
      <xsl:otherwise>
        <xsl:if test="(ancestor::*)[2]/@rowheight">
          <xsl:attribute name="block-progression-dimension">
            <xsl:value-of select="(ancestor::*)[2]/@rowheight"/>
          </xsl:attribute>
        </xsl:if>
      </xsl:otherwise>
    </xsl:choose>
    <xsl:apply-templates />
  </xsl:element>
</xsl:template>

<xsl:template match="th">
  <fo:table-cell xsl:use-attribute-sets="table.data.th">
```

```
    <xsl:call-template name="cell-span"/>
    <xsl:if test="@valign">
      <xsl:attribute name="display-align">
        <xsl:value-of select="@valign"/>
      </xsl:attribute>
    </xsl:if>
    <fo:block >
      <xsl:if test="@align">
        <xsl:attribute name="text-align">
          <xsl:value-of select="@align"/>
        </xsl:attribute>
      </xsl:if>
      <xsl:apply-templates />
    </fo:block>
  </fo:table-cell>
</xsl:template>

<xsl:template match="td">
  <fo:table-cell xsl:use-attribute-sets="table.data.td">
    <xsl:call-template name="cell-span"/>
    <xsl:if test="@valign">
      <xsl:attribute name="display-align">
        <xsl:value-of select="@valign"/>
      </xsl:attribute>
    </xsl:if>
    <fo:block >
      <xsl:if test="@align">
        <xsl:attribute name="text-align">
          <xsl:value-of select="@align"/>
        </xsl:attribute>
      </xsl:if>
      <xsl:apply-templates />
    </fo:block>
  </fo:table-cell>
</xsl:template>

<xsl:template name="cell-span">
  <xsl:if test="@colspan">
    <xsl:attribute name="number-columns-spanned">
      <xsl:value-of select="@colspan"/>
    </xsl:attribute>
  </xsl:if>
  <xsl:if test="@rowspan">
    <xsl:attribute name="number-rows-spanned">
      <xsl:value-of select="@rowspan"/>
  </xsl:attribute>
  </xsl:if>
</xsl:template>
```

The stylesheet looks long, though, it is only to map the element name of the SampleDoc to that of XSL formatting object. Below shows the points.

1. Layout attributes defined for the table element is set for fo:table layout property to control the autmatic table layouts (the value auto or fixed is varid). [12]

[12]Antenna House XSL Formatter V1.1 dosen't support auto table formatting. Even if it is speicfied as "auto", the layout will be fixed.

2. The width attribute defined for the table element is set for inline-progression-dimension property of fo:table object as the width of the entire table.
3. The number, width attributes defined for the col element is set for column-number, column-width property of the fo:table-column object. Then, the column width can be specified. It's possible to specify the column width not by the fixed number but by % value. There are various ways to specify parameter, such as specifying an absolute value to the width attribute of the table element, or specifying the width of the table column as % value. (13)
4. The row height attribute defined for the table element is set to block-progression-dimension of fo:table-row object. The attribute refer to the template that process tr element, so it's specified as "(ancestor::*)[2]/@rowheight"
5. The height attribute defined for the tr element is set for block-progression-dimension of the fo:table-row object. It is also set to precede the rowheight attribute defined for the table element.
6. The colspan, rowspan attributes defined for the th, td elements are set to each number-columns-spanned, number-rows-spanned of fo:table-cell object. Then, cells can be spanned in this way. In addition, the align, valign attributes are set to text-align property, display-align property of fo:table-cell in fo:block. In this way the text in the cell can be aligned.

(13)Furthermore, there is a way to specify an absolute value for a part of the columns, while, specify proportional-column-width() function to the rest of the columns and share width of the cells proportionally.

Example of Table Construction

Below is an example of table construction.

When nothing are specified. the width of the column is devided by ther number of the column equally.

Symbol	How to read	Meaning
|	vertical bar	Means one element or another is to be used.
?	question mark	Means that the element appears zero or one time.
,	comma	Means that elements appear in the same order in that element inside the document.
*	asterisk	Means that the element appears zero or more times.
+	plus	Means that the element appears one or more time.
()	parentheses	Means to group plural numbers of elements in the parentheses.
	empty	means that only one element can be described.

Specify the column width by the col element. Set 10%, 20%, 40% from the left. Specify height="1em" in the tr element in order to set the height 1em only for the table header. Contents in the table header cells are center-aligned. Contents in the first and the second columns from the left are center aligned, center valigned by specifying valign="center" align="center"

Symbol	How to read	Meaning
|	Vertical bar	Means one element or another is to be used.
?	question mark	Means that the element appears zero or one time.
,	comma	Means that that elements appear in the same order in that element inside the document.
*	asterisk	Means that the element appears zero or more times.
+	plus	Means that the element appears one or more time.
()	parentheses	Means to group plural numbers of elements in the parentheses.
	empty	Only one element can be described.

Processing list Elements

Output specification

- The list elements (ol,ul,dl) of the sampleDoc is formatted as the table that the label and the body of the list are lined in horizontal mode.
- But, the dt element also make it possible to be formatted such as the label and the body of the list are lined in vertical mode. (14)
- The ordered list element (ol) make it possible to take various types of numbers that are set in the labels.
- The unordered list elements (ul) make it possible to specify various types of bullets that are set in the labels.

Comparing the List of SampleDoc with the List of XSL

In order to put the list elements to both the label of the list and body of the list, it is neccessaly to transform them to fo:list-block in XSL. Let us compare lists of SampleDoc with those of XSL formatting objects.

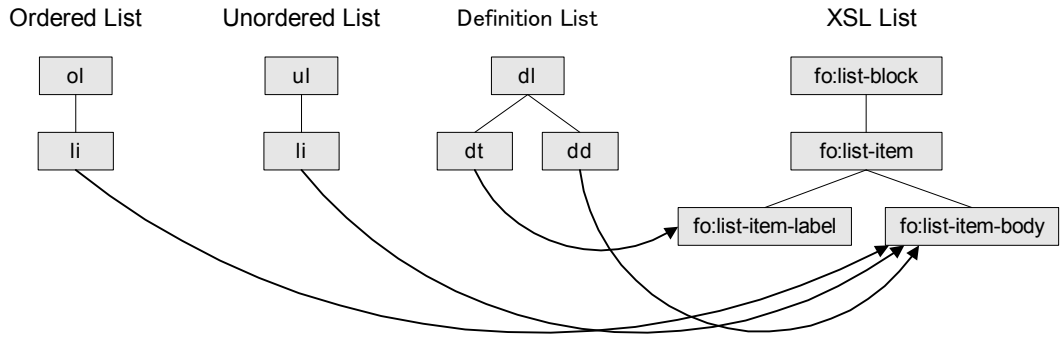
Element	Meaning	Definition
ol	ordered list	(li)*
ul	unordered list	(li)*
li	list item	(a group of inline elements)*
dl	definition list	(dt, dd)*
dt	definition term	(a group of inline elements)*
dd	definition details	(a group of inline elements)*

While, the list of XSL is as follows:

Element	Meaning	Definition
list-block	The formatting object that format the lists	(list-item+) provisional-distance-between-starts：Specify the distance between the start indent of the fo:list-item-label and the start indent of the fo: list-item-body. provisional-label-separation：Specify the distance between the end of the list-item-label and the start of the list item body.
list-item	Express one list item, including list label and list body.	(list-item-label,list-item-body)
list-item-label	Express the label of a list item.	(%block;)+
list-item-body	Express the body of a list item.	(%block;)+

Shown below is the structure of the both lists

(14)This is the same desplay format as the browser format of dt in HTML.



The lists of SampleDoc and XSL

The followings are remarkable diferences between two.

- The bullets in the label are autmatically generated by the ul,ol elements in HTML. In XSL, they have to be generated within the list-item-label region using a stylesheet.
- In order to transform the definition list element into a fo:list-block object, set the dt content in the fo:list-item-label and the dd content in the fo:list-item-body.
- There is no function to calculate the width of the fo:list-item-label formatting object. An appropriate value must be specified in the XSLT stylesheet.

Each of the XSLT stylesheet is shown below:

Templates that Process the Ordered List.

The ol templates for ordered list

```
<xsl:param name="list-startdist-default" select="string('2em')"/>
<xsl:param name="list-gap-default" select="string('0.5em')"/>

<xsl:attribute-set name="list.item" >
  <xsl:attribute name="space-before">0.4em</xsl:attribute>
  <xsl:attribute name="space-after">0.4em</xsl:attribute>
  <xsl:attribute name="relative-align">baseline</xsl:attribute>
</xsl:attribute-set>

<xsl:template match="ol">
  <!-- determine the distance between the start  of the list-item-label and the start
of the list-item-body, the distance between the end of the list-item-label and the
start of the list-item-body. -->
  <xsl:variable name="start-dist-local">
    <xsl:choose>
      <xsl:when test="./@startdist">
        <xsl:value-of select="./@startdist"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="$list-startdist-default"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:variable>

  <xsl:variable name="gap-local">
    <xsl:choose>
```

```
      <xsl:when test="./@gap">
        <xsl:value-of select="./@gap"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="$list-gap-default"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:variable>


  <!-- generate fo:list-block -->
  <fo:list-block provisional-distance-between-starts="{$start-dist-local}"
provisional-label-separation="{$gap-local}" >
    <!-- Process the descendants of li -->
    <xsl:apply-templates />
  </fo:list-block>
</xsl:template>

<xsl:template match="ol/li">
  <fo:list-item  xsl:use-attribute-sets="list.item">
    <!-- generate list-item-label-->
    <!-- the end position of the list-item-label is calculated by label-end()
function -->
    <!-- label format is specified in the type attribute. The initial value is '1'.-->
    <fo:list-item-label end-indent="label-end()">
      <fo:block text-align="end">
        <xsl:choose>
          <xsl:when test="../@type">
            <xsl:number format="{../@type}"/>
          </xsl:when>
          <xsl:otherwise>
            <xsl:number format="1."/>
          </xsl:otherwise>
        </xsl:choose>
      </fo:block>
    </fo:list-item-label>
       <!-- generate the list-item-body -->
    <!-- The start position of the list-item-label is calculated by body-start()
function -->
    <fo:list-item-body start-indent="body-start()"  text-align="justify" >
      <fo:block>
        <!-- the descendants of li are specified by the descendants of templates. -->
        <xsl:apply-templates/>
      </fo:block>
    </fo:list-item-body>
  </fo:list-item>
</xsl:template>
```

Specify the positions of the label and the body.

When formatting a list, it is important to arrange the label and the body. In the fo:list-block:

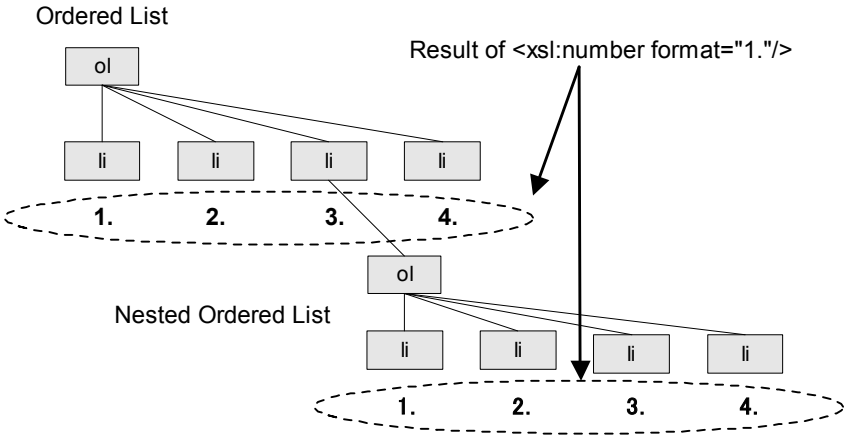
- provisional-distance-between-starts specifies the distance between the start of the list-item-label and the start of the list-item-body
- provisional-label-separation specifies the distance between the end of the list-item-label and the start of the list-item-body.

The template specifies these to apply to the startdist, gap attributes of the ol element. But actually the descendants of the fo:list-item are formatted. In order to specify the position of the end of the list-item-label and the start of the list-item-body, the label-end(), body-start() functions are used in the template. [15] These two functions count the position by refering the values of provisional-distance-between-starts, provisional-distance-between-starts specified in fo:list-block. [16]

These function specification is just a manner. If you understand how to layout lists, it may be no problem to apply these mechanically.

Label format

The list label consists of a sequence of numbers. A sequence of numbers is generated by the default <xsl:number format="1."/>. xsl:number counts the same level (sibling) of the li elements in the XML source document, returns a sequence number of the current li elements. Therefore li elements are processed correctly eve if the lists are nested.



Format the ordered list label processed by xsl:number

[15] These are called 'Proterty Value Function' in the XSL Specification.

[16] label-end() = the list item width of fo:list-block - (provisional-distance-between-starts + start-indent of the label - provisional-label-separation)
body-start() =start-indent of the label + provisional-distance-between-starts

Form of a sequence of numbers is specified by the type attribute of the ol element. An ordered list can take various kinds of types.

Form	Output
1	1,2,3,4...
01	01,02,03,04...
a	a,b,c,d,...x,y,z,aa,ab,ac...
A	A,B,C,D,...X,Y,Z,AA,AB,AC...
ア	ア,イ,ウ,エ...
あ	あ,い,う,え...
イ	イ,ロ,ハ,ニ...
一	一,二,三,四...
壱	壱,弐,参,四...

The format attribute specifies the characters that represent zero, 1 in UNICODE, and other parentheses. As the format of label can be specified in the XML source document, the document can be made flexibly.

Example of ordered list

XML source data

```
<ol type="a.">
  <li>kind of list
    <ol type="一.">
      <li>unordered list</li>
      <li>ordered list</li>
      <li>definition list</li>
    </ol>
  </li>
  <li>list element
    <ol>
      <li>row</li>
      <li>column</li>
      <li>cell</li>
    </ol>
  </li>
  <li>block element and inline element</li>
</ol>
```

Following is the output result.

a. kind of list
   一. unordered list
   二. ordered list
   三. definition list
b. list element
   1. row
   2. column
   3. cell
c. block element and inline element

Templates that Process the Unordered List

Templates of unordered list

```
<xsl:param name="list-startdist-default" select="string('2em')"/>
<xsl:param name="list-gap-default" select="string('0.5em')"/>
<xsl:attribute-set name="list.item" >
  <xsl:attribute name="space-before">0.4em</xsl:attribute>
  <xsl:attribute name="space-after">0.4em</xsl:attribute>
  <xsl:attribute name="relative-align">baseline</xsl:attribute>
</xsl:attribute-set>

<xsl:template match="ul">
  <!-- determine the distance between the start of the list-iten-label and the start
of the list-item-body, the distance between the end of the list-item-label and the
start of the list-item-body. -->
  <xsl:variable name="start-dist-local">
    <xsl:choose>
      <xsl:when test="./@startdist">
        <xsl:value-of select="./@startdist"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="$list-startdist-default"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:variable>

  <xsl:variable name="gap-local">
    <xsl:choose>
      <xsl:when test="./@gap">
        <xsl:value-of select="./@gap"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="$list-gap-default"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:variable>

  <!-- Generate fo:list-block. -->
  <fo:list-block provisional-distance-between-starts="{$start-dist-local}"
provisional-label-separation="{$gap-local}" >
    <!-- Process the descendants of li -->
    <xsl:apply-templates/>
  </fo:list-block>
</xsl:template>

<xsl:template match="ul/li">
  <fo:list-item xsl:use-attribute-sets="list.item" >
      <!-- Generate list label.-->
      <!-- The end position of the label is calculated by label-end()function. -->
      <!-- The characters for label of line are specified in the type attribute.
Initial value is「・」 -->
    <fo:list-item-label end-indent="label-end()">
      <fo:block text-align="end">
        <xsl:choose>
          <xsl:when test="../@type='disc'">
            <xsl:text>●</xsl:text>
          </xsl:when>
          <xsl:when test="../@type='circle'">
```

```
            <xsl:text>o</xsl:text>
          </xsl:when>
          <xsl:when test="../@type='square'">
            <xsl:text>□</xsl:text>
          </xsl:when>
          <xsl:when test="../@type='bsquare'">
            <xsl:text>■</xsl:text>
          </xsl:when>
          <xsl:otherwise>
            <xsl:text>・</xsl:text>
          </xsl:otherwise>
        </xsl:choose>
      </fo:block>
    </fo:list-item-label>
    <!-- Generate the list body.-->
    <!-- The starting position of the label is calculated by the body-start()
function -->
    <fo:list-item-body start-indent="body-start()" text-align="justify" >
      <fo:block>
        <xsl:apply-templates/>
      </fo:block>
    </fo:list-item-body>
  </fo:list-item>
</xsl:template>
```

Specify characters for label of line

The difference between the ol and ul templates is only label processing. In the case of unordered list, the characters are set in the label. Types of characters can be specified in the type attribute in the ul element.

This is an example of the template that place an image as a character for label of line. Specify the image file by img:file name in the type attribute defined for the ul element.

The template that use a image as a character for label of line.

```
<!-- The template that use a image as a character for label of line.-->
<xsl:template match="ul[substring(@type,1,4)='img:']/li">
  <fo:list-item  xsl:use-attribute-sets="list.item" >
    <fo:list-item-label end-indent="label-end()">
      <fo:block text-align="end">
        <fo:external-graphic src="{substring-after(../@type,substring(../
@type,1,4))}" content-height="1.2em" content-width="1.2em"/>
      </fo:block>
    </fo:list-item-label>
    <fo:list-item-body start-indent="body-start()" text-align="justify" >
      <fo:block>
        <xsl:apply-templates/>
      </fo:block>
    </fo:list-item-body>
  </fo:list-item>
</xsl:template>
```

An example of unordered list

XML source data

```
<ul type="square">
  <li>type of list
    <ul type="disc">
      <li>unordered list</li>
      <li>ordered list</li>
      <li>definition list</li>
    </ul>
  </li>
  <li>table element
    <ul>
      <li>row</li>
      <li>column</li>
      <li>cell</li>
    </ul>
  </li>
  <li> block element and inline element</li>
</ul>
```

Folowing is the output result

- kinds of list
  - unordered list
  - orered list
  - definition list
- list element
  - row
  - column
  - cell
- block element and inline element

XML source data

```
<ul type="img:bullet-leaf.png">
  <li>type of list
    <ul type="img:bullet-star.png">
      <li>unordered list</li>
      <li>ordered list</li>
      <li>definition list</li>
    </ul>
  </li>
  <li>list element
    <ul type="img:bullet-blue-circle.png">
      <li>row</li>
      <li>column</li>
      <li>cell</li>
    </ul>
  </li>
  <li>block element and inlinte element</li>
</ul>
```

Folowing is the output result

- type of list
  - unordered list
  - ordered list
  - definition list
- list element
  - row
  - column
  - cell
- block element and inline element

Templates that Process the Definition List

The following problem occures when transforming the definition list into fo:list-block formatting object.

- In the SampleDoc specification, child of dl element is supposed to appear in dt, dd order (dt,dd)*, but in HTML the condition is more relaxed. The pattern of only dt, only dd also exist. **<!ELEMENT DL (DT|DD)+ >**
- dt can be mapped to fo:list-item-label, dd can be mapped to fo:list-item-body. But there are no tags to be mapped to generate fo:list-item in the XML source document.

It is better to be able to process definition list like HTML also in XSL. But the above two problem cannot be solved by the data driven stylesheet which process the tags in the XML source document in appearing order. It is necessary to find a pair of dt, dd in the XML source document. Following is the template that realize this process. The template is a remodel of the sample XSLT stylesheet described in the XSL specification.

Templates of definition list

Templates of definition list

```
<xsl:attribute-set name="dt" >
  <xsl:attribute name="font-weight">bold</xsl:attribute>
</xsl:attribute-set>

<xsl:attribute-set name="dd.list" >
  <xsl:attribute name="space-before">0.3em</xsl:attribute>
  <xsl:attribute name="space-after">0.5em</xsl:attribute>
</xsl:attribute-set>

<xsl:attribute-set name="dd.block" use-attribute-sets="dd.list">
  <xsl:attribute name="start-indent" >inherit + 4em</xsl:attribute>
</xsl:attribute-set>

<xsl:template match="dl">
  <xsl:choose>
    <xsl:when test="@type='list'">
      <xsl:call-template name="dl.format.list" />
    </xsl:when>
    <xsl:otherwise>
      <xsl:call-template name="dl.format.block" />
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

```
<xsl:template name="dl.format.block">
  <xsl:apply-templates />
</xsl:template>

<xsl:template name="dl.format.list">
  <xsl:variable name="start-dist-local">
    <xsl:choose>
      <xsl:when test="./@startdist">
        <xsl:value-of select="./@startdist"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="$dl-startdist-default"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:variable>

  <xsl:variable name="gap-local">
    <xsl:choose>
      <xsl:when test="./@gap">
        <xsl:value-of select="./@gap"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="$dl-gap-default"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:variable>

  <fo:list-block provisional-distance-between-starts="{$start-dist-local}"
provisional-label-separation="{$gap-local}" >
    <xsl:call-template name="process.dl.list"/>
  </fo:list-block>
</xsl:template>

<xsl:template name="process.dl.list">
  <xsl:param name="dts" select="/.."/>
  <xsl:param name="dds" select="/.."/>
  <xsl:param name="nodes" select="*"/>

  <xsl:choose>
    <xsl:when test="count($nodes)=0">
      <!--  data end: Process the elements stocked in dts, dds. -->
      <xsl:if test="count($dts)>0 or count($dds)>0">
        <fo:list-item  xsl:use-attribute-sets="list.item">
          <fo:list-item-label end-indent="label-end()">
            <xsl:apply-templates select="$dts"/>
          </fo:list-item-label>
          <fo:list-item-body start-indent="body-start()">
            <xsl:apply-templates select="$dds"/>
          </fo:list-item-body>
        </fo:list-item>
      </xsl:if>
    </xsl:when>

    <xsl:when test="name($nodes[1])='dd'">
      <!-- dd is stored in dds fnction, call itself recursively.-->
      <xsl:call-template name="process.dl.list">
        <xsl:with-param name="dts" select="$dts"/>
        <xsl:with-param name="dds" select="$dds|$nodes[1]"/>
        <xsl:with-param name="nodes" select="$nodes[position()>1]"/>
```

```
      </xsl:call-template>
    </xsl:when>

    <xsl:when test="name($nodes[1])='dt'">
      <!-- Process the elements stocked in dts, dds. -->
      <xsl:if test="count($dts)>0 or count($dds)>0">
        <fo:list-item  xsl:use-attribute-sets="list.item">
          <fo:list-item-label end-indent="label-end()">
            <xsl:apply-templates select="$dts"/>
          </fo:list-item-label>
          <fo:list-item-body start-indent="body-start()">
            <xsl:apply-templates select="$dds"/>
        </fo:list-item-body></fo:list-item>
      </xsl:if>
      <!-- dt is stored in dts variable, call itself recursively.-->
      <xsl:call-template name="process.dl.list">
        <xsl:with-param name="dts" select="$nodes[1]"/>
        <xsl:with-param name="nodes" select="$nodes[position()>1]"/>
      </xsl:call-template>
    </xsl:when>

    <xsl:otherwise>
      <!-- The children of dl must be only dt, dd. -->
      <!-- Unfortunately, xsl:message does not work in MSXML3. -->
      <xsl:message>
        <xsl:text>
         The elements except dt,dd are specified as a child of dl element.
        </xsl:text>
        <xsl:value-of select="name($nodes[1])"/>
        <xsl:text>).</xsl:text>
      </xsl:message>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match="dt">
  <xsl:element name="fo:block" use-attribute-sets="dt">
    <xsl:if test="../@mode='debug'">
      <xsl:attribute name="border-color">blue</xsl:attribute>
      <xsl:attribute name="border-style">dashed</xsl:attribute>
      <xsl:attribute name="border-width" >thin</xsl:attribute>
    </xsl:if>
    <xsl:apply-templates />
  </xsl:element>
</xsl:template>

<xsl:template match="dd">
  <xsl:choose>
    <xsl:when test="../@type='list'">
      <xsl:element name="fo:block" use-attribute-sets="dd.list">
        <xsl:if test="../@mode='debug'">
          <xsl:attribute name="border-color">red</xsl:attribute>
          <xsl:attribute name="border-style">solid</xsl:attribute>
          <xsl:attribute name="border-width" >thin</xsl:attribute>
        </xsl:if>
        <xsl:apply-templates />
      </xsl:element>
    </xsl:when>
    <xsl:otherwise>
```

```
      <xsl:element name="fo:block" use-attribute-sets="dd.block">
        <xsl:if test="../@mode='debug'">
          <xsl:attribute name="border-color">red</xsl:attribute>
          <xsl:attribute name="border-style">solid</xsl:attribute>
          <xsl:attribute name="border-width" >thin</xsl:attribute>
        </xsl:if>
        <xsl:apply-templates />
      </xsl:element>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

In this stylesheet, you have to determine which transformation to use from the following type by the type attribute defined for the dl element:

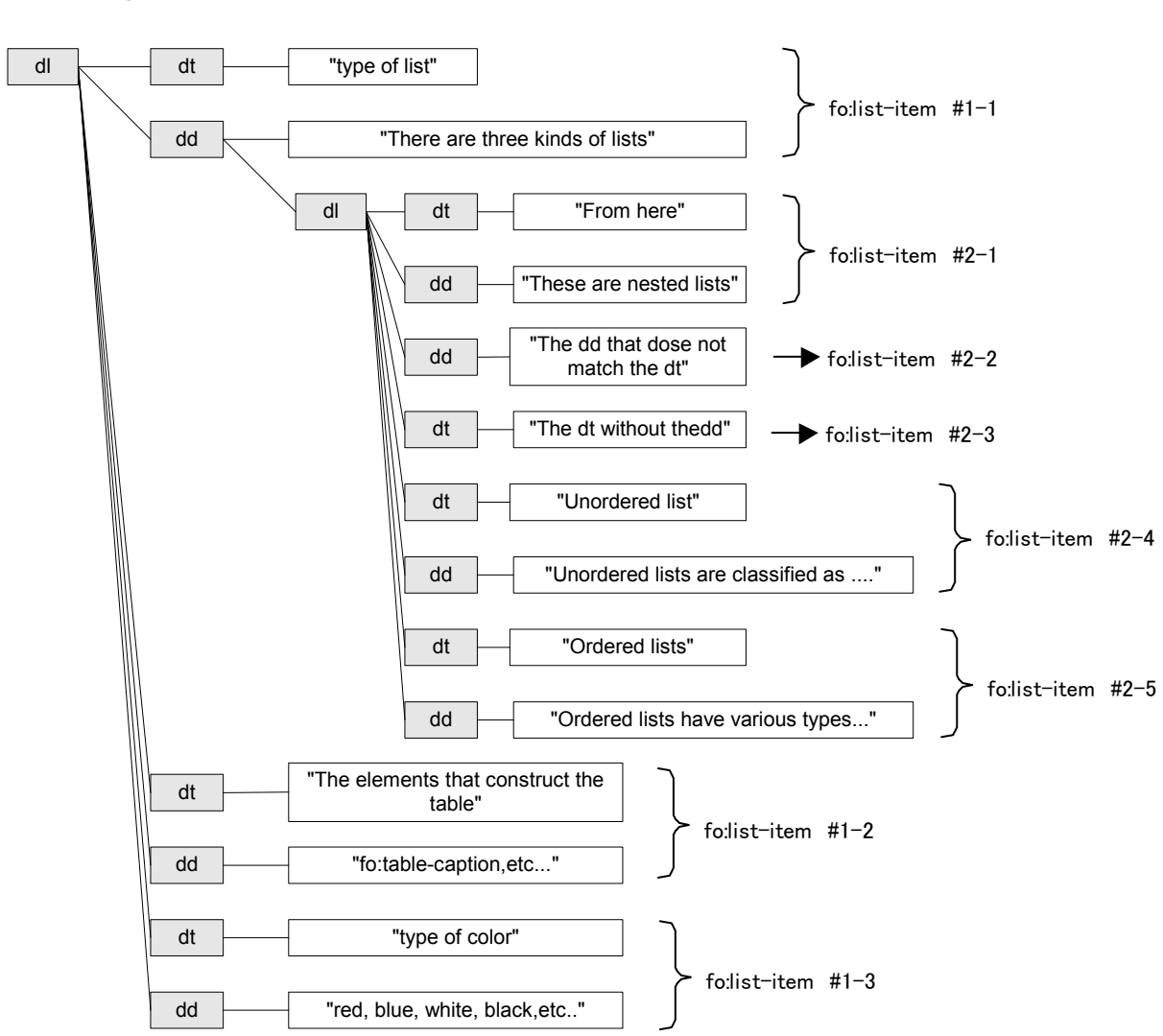
- list type: the label (dt) and the body(dd) are lined in horizontal way as fo:list-block.
- HTML type: the label (dt) and the body (dd) are lined in vertical way.

When "list" is specified in the type attribute, select the first one.

In case of the list type, after generating fo:list-block, the process.dl.list template will process these. The process.dl.list template processes the following:

1. The process.dl.list template take descendant elements of dl, start processing from the first element.
2. After getting dt, dd, store each of them to the dts, dds variable in appearing order.
3. When getting the next dt, take out dt, dd that are stored to dts, dls. Output fo:list-item and its descendants. Initialize the dts, dds, and store the dt to dts variable.
4. The above two processes are repeated until all the dt, dd are taken out.
5. It is the end of the process when all the dt, dd under dl, are processed. If there are still dt, dd left in the dts, dds variable, take out these and output fo:list-item and its descendants.

The following shows how this fo:list-item is generated from the dt, dl. fo-list-item #1-n is generated from the first dl. #2-n is generated from the nested dl. The figure shows the contents of XML source document described later.



Transforming the definition list into the list type.

The process.dl.list template calls itself recursively. In the general programing language, it is natural to assign a value to a valiable and process using a valiable. But in XSLT, it is impossible to assign a value to a variable, but it is possible to initiarize a value instead. Loops are formed by taking recursive process.

For HTML type, the dl.format.block template transforms. Only the template have to do is to call the descendant templates in order to process dt, dd in apearing order.

Example of a definition list

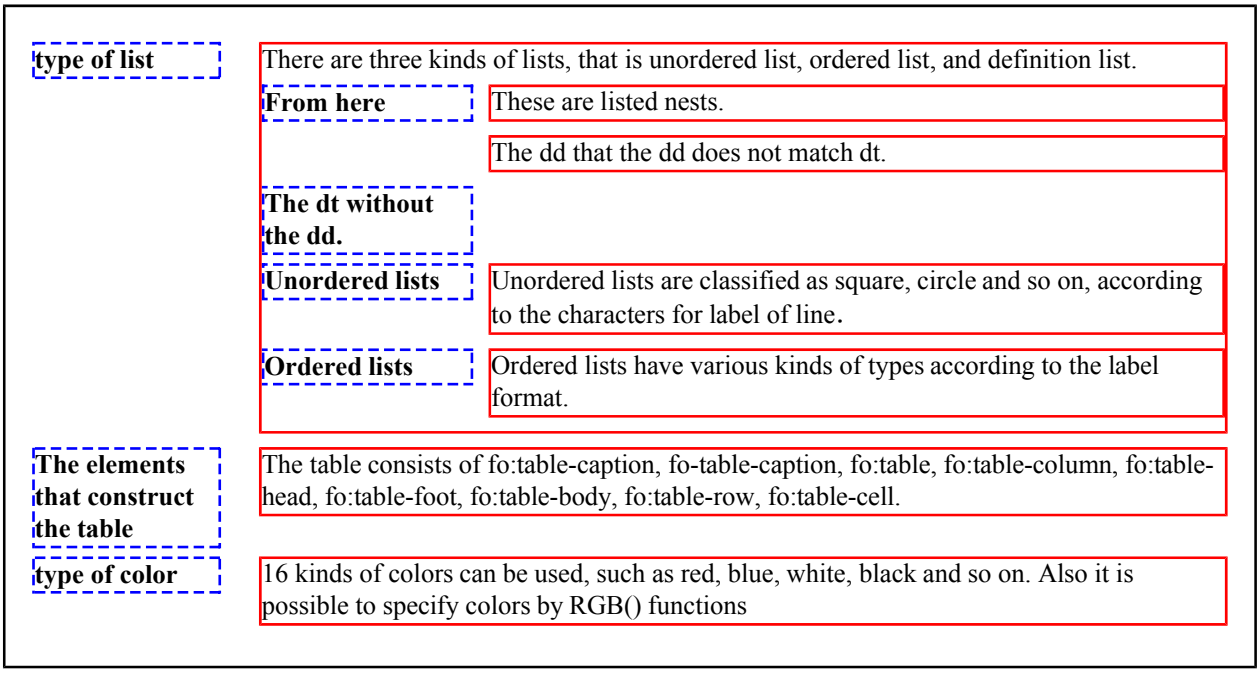
The following is a sample source data

A sample data of the definition list

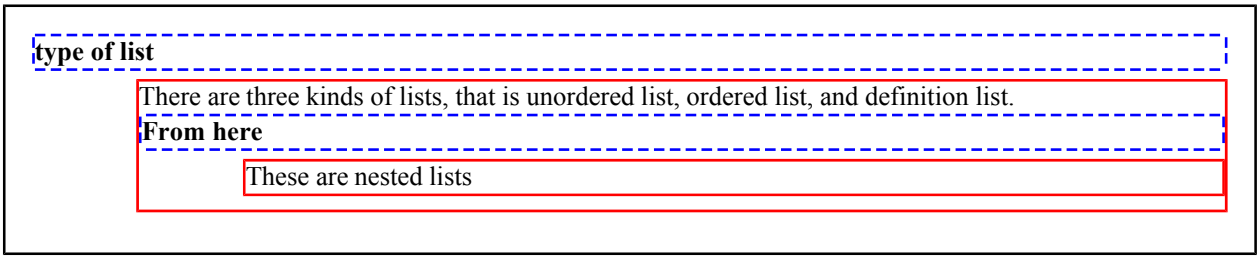
```
<dl>
  <dt>Type of lists</dt>
    <dd>There are three kinds of lists, that is unordered list, ordered list, and
```

```
definition list.
    <dl class="list">
      <dt>From here</dt>
      <dd>These are nested lists</dd>
      <dd>The dd that does not match dt. </dd>
      <dt>The dt without the dd</dt>
      <dt>unordered list</dt>
      <dd>Unordered lists are classified as square, circle and so on, according to
the characters for label of line. </dd>
      <dt>Ordered lists</dt>
      <dd>Ordered lists have various kinds of types according to the label format.</
dd>
    </dl>
  </dd>
  <dt>The elements that construct the table</dt>
  <dd>fo:The table consists of table-caption, fo-table-caption, fo:table, fo:table-
column, fo:table-head, fo:table-foot, fo:table-body, fo:table-row, fo:table-cell</dd>
  <dt>kinds of color</dt>
  <dd>16 kinds of colors can be used, such as red, blue, white, black and so on. Also
it is possible to specify colors by RGB() functions.</dd>
</dl>
```

The following is the output of list type transformation. To make it look better, fo:list-item-labeland fo:list-item-body are bordered.



The following is the output of HTML type transormation.



The dd that the dt does not match dt.

The dd without the dt.

Unordered lists

Unordered lists are classified as square, circle and so on, according to the characters for label of line.

Ordered lists

Ordered lists have various kinds of types according to the label format.

The elements that construct the table

The table consists of fo:table-caption, fo-table-caption, fo:table, fo:table-column, fo:table-head, fo:table-foot, fo:table-body, fo:table-row, fo:table-cell.

type of color

16 kinds of colors can be used, such as red, blue, white, black and so on. Also it is possible to specify colors by RGB() functions.

Appendix

The folowing are the books referring to XSL, XSLT.

XSLT Programmer's Reference	Michael Kay / WROX This is a manual of XSLT. This book has rich contents. The second edition was issued in april, 2001.
PROFESSIONAL XSL	Kurt Cagle / WROX This is a manual of XSL in general. The contents have various fields, such as XSLT, XSL-FO, CSS, SVG. XSL Formatter, our products, is introduced in the chapter 9, CSS and XSL-FO using three pages.
XSL Companion	Neil Bradley / Addison-Wesley This book explains about XSLT, XSL-FO, HTML briefly in general. XSL-FO is explained a lot, but it is somewhat old seen from the current recommendation.
PureSmartDoc	For more information about PureSmartDoc, please refer to: http://www.asahi-net.or.jp/~dp8t-asm/java/tools/

Antenna House Web site: http://www.antennahouse.com