How to Develop Stylesheet for XML to XSL-FO Transformation

July, 2005
2nd Edition

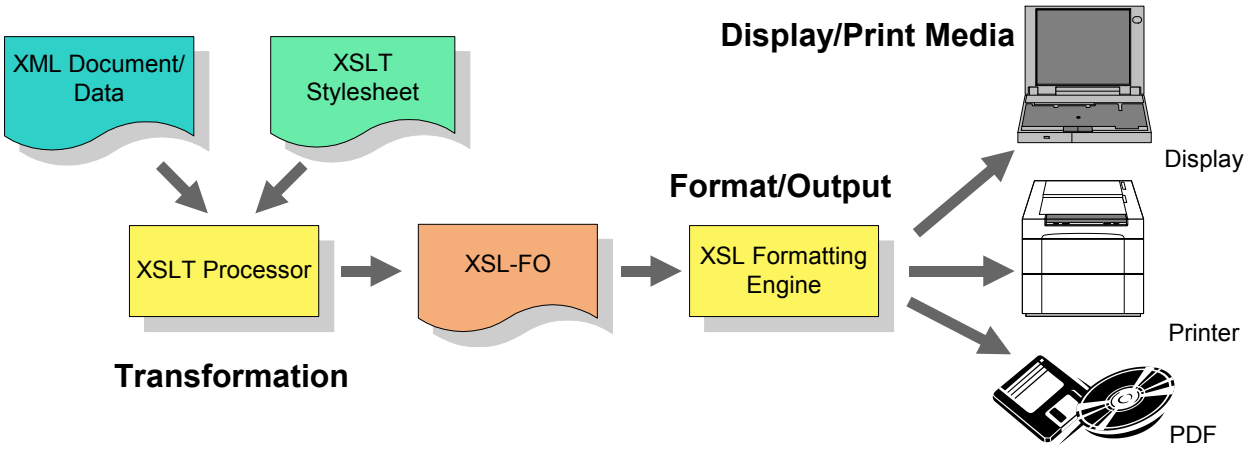
Antenna House, Inc.

Table of Contents

Preface

Extensible Stylesheet Language (XSL) (Source Information [1]) has been brought to the attention of a wide audience as a specification recommended by W3C on 15th October, 2001 for displaying and printing the XML document. The following is the general process to transform the XML document into XSL Formatting Objects (XSL-FO) and print it.

1. Develop the stylesheet that conforms to the DTD of the source XML document to create the target output.
2. Input the XML document and the XSL stylesheet to the XSLT processor to create XSL-FO.
3. Get the target outputs such as paper-based output, PDF output, 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 stylesheet. The knowledge about XSLT and XSL is necessary to develop XSLT stylesheet. As for XSLT, many reference books has been published other than the specification (Source Information [2]) . Probably, XSLT is already much more familiar one, since it is often used for the conversion from XML to HTML. The XSL specification (Source Information [1]) 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 stylesheet by knowing some regular contents and patterns.

This report explains how to edit stylesheet which is used for transforming XML documents into XSL-FO according to the example of SimpleDoc. SimpleDoc is made for this report as a format to write a simple document. This is based on PureSmartDoc (Source Information [4]) presented by Tomoharu Asami. To make it a simple, the number of the elements is reduced and the useful functions for writing and formatting documents are added.

This report explains how to edit a stylesheet which is used for transforming a SimpleDoc document into XSL-FO. This report itself is the instance XML document of SimpleDoc.dtd, and it is then ready to be formatted by Antenna House XSL Formatter with the stylesheet which transforms the SimpleDoc document explained here into XSL-FO.

Step for XSL-FO Transformation

Now, what steps are necessary to develop XSL stylesheet? 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 types of elements, contents of elements, appearing order of elements and values of properties are necessary for developing 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 sizes 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 simple stylesheet.
Develop an XSL stylesheet	Put the instructions to the stylesheet in order to transform XML source documents into the target printing form. Map the XML source document to XSL formatting objects that can generate the output specification. The stylesheet have the similar aspect as the general programming languages, while it may be difficult if you do not understand the feature of the XSL. (1)

(1) Refer to the definition list template in this report. In XSL, 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.

SimpleDoc Organization

The following table shows the structure of SimpleDoc treated in this report. For more detail, refer to SimpleDoc.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 property.
ul	unordered list	(li*) Specify a character for label of line by the type property.
ol	ordered list	(li*) Specify format of number in the label by the type property.
dl	definition list	(dt, dd)* Specify whether to format the block in horizontal way or in vertical way by the type property.
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 property. Specify the width of the entire table by the width property.
col	column format	EMPTY Speciry the number of the column by the number property, the width of the column by the width property.
thead	table header	(tr*)
tfoot	table footer	(tr*)
tbody	table body	(tr*)
tr	table row	(th | td)* Specify the height of the row by the height property.

Element	Meaning	Definition
th	table header	(#PCDATA | a group of inline elements | a group of block elements)* Specify the number of the columns to be expanded across, the number of the rows to be expanded vertically by the colspan and rowspan properties. The align property allows horizontal alignment to be set to left, right, or center. The valign property 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 across, the number of the rows to be expanded vertically by the colspan and rowspan properties. The align property allows horizontal alignment to be set to left, right, or center. The valign property 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 property.
a	anchor(link)	(#PCDATA | a group of inline elements)* Specify URL as the value of href property.
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 features of SimpleDoc are:

- You can start writing a document from part, also start from section. It has a flexible structure so that it can map to various kinds of documents.
- The number of the block element and in-line element are reduced to a minimum amount. The div in the block element and the span in the in-line element are defined to give various extensions.
- The br element is defined 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.
- The problem is that the content and style is mixed.

Hello! World

Simple Example of Transforming SimpleDoc into XSL-FO

Now, we show the simplest simple stylesheet that transforms SimpleDoc to XSL-FO as follows.

Source XML Document (Hello.xml)

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

XSL Stylesheet(Simple.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-reference="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>

<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-reference="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">SimpleDoc</fo:inline>
        </fo:block>
      </fo:block>
    </fo:flow>
  </fo:page-sequence>
</fo:root>
```

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

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

Stylesheet Structure

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

- A stylesheet is a set of templates. The descendant of the root element, xsl:stylesheet consists of xsl:template elements. Each xsl:template is applied so that the xxx tag of the source XML document may be processed by 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 recursively, continues until the processor returns to the root element and there are no more templates to be processed.

Block Elementand Inline Element

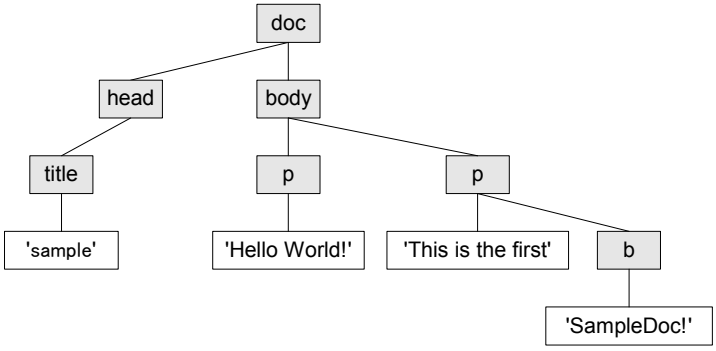
Please note how the XSL stylesheet maps block elements and inline elements in source element to formatting objects.

- 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 document 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 normally 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.

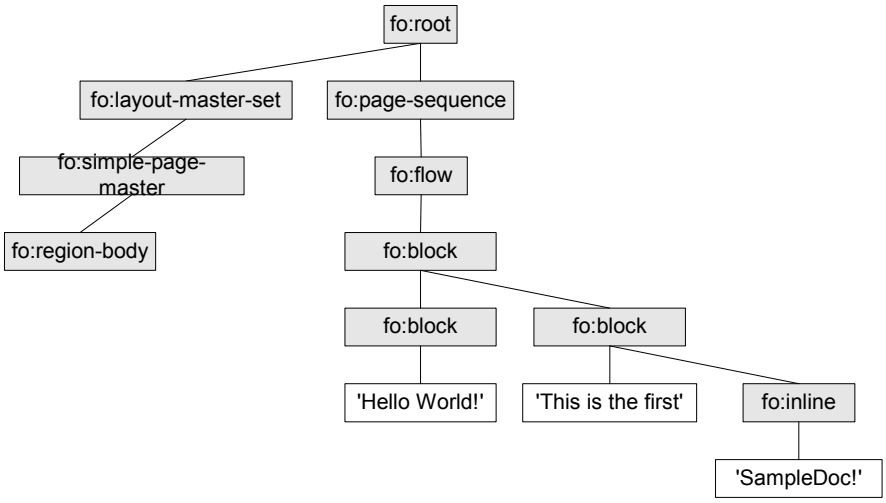
XSL-FO Tree Structure

Next, please pay attention to the XSL-FO tree structure. The following illustrates the structure of the XML document.



Hello.xml Tree Structure

In contrast, the tree structure of XSL-FO is as follows. 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 arranged in in pages.



XSL-FO Tree after XSL Processing

fo:layout-master-set that defines the page layouts should precede fo:page-sequence（preceding-sibling）that is an actual content of pages. XSL 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 transferred as it used to be, using <xsl:template match="xxx">...<xsl:apply-templates;> The result has the same tree structure. In the Simple.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 omitted according to the instruction of <xsl:apply-templates select="body" />

Developing a Practical Stylesheet

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
Organization	cover, table of contents, the text, and index are proceseed in order from the top.
Header, Hooter	Specify the header, footer regions. Do not set text inside the whitespace on the both side of the body.

【Cover/Table of Contents】

Item	Specification
Margin of paper	Top: 25mm, Bottom: 25mm, Left: 25mm, Right: 25mm

【Body Region 】

Item	Specification
Margin	Margin top: 20mm, bottom: 20mm, left: 0mm, right: 0mm
Content	Consists of head, table, list, paragraph and image.
Writing mode	lr-tb
Column	1
Default font size	10pt
Text align	justify
Other conditions	Place a header region and a footer region. The content of the footer region is fore-edge justified and changed by right and left pages. 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.

【 Header Region 】

Item	Specification
Extent	10mm
Writing Mode	lr-tb
Content	Print the title Font size is 9pt. Center aligned in the inline progression direction, bottom aligned in the block progression direction. Create an index on the upper part of a page.

【 Footer Region 】

Item	Specification
Extent	10mm
Writing Mode	lr-tb
Contents	Print a page number and a sub-title of the current page on the fore-edge side.

【Index】

Item	Specification
Margin	Margin top: 25mm, bottom: 25mm, left: 25mm, right: 25mm
Column	2
Column rule	20mm

XSL Stylesheet Organization

XSL Stylesheet consists of the following 5 files.

File Name	Contents/Usage
SD2FO-DOC.XSL	Main body of the XSL Stylesheet
attribute.xsl	The file which defines the property of XSL-FO collectively.
param.xsl	The file which defins values, such as paper size, as a parameter.
index.xsl	The file which defines the processing of creating an index collectively.
article.xsl	The stylesheet for a document format without a cover, a table of contents and an index.

SD2FO-DOC.XSL consists of the following top level XSL elements.

XSL elements	Contents/Usage
xsl:include	Includes the stylesheets divided according to the function.
xsl:param	Specifies the value of paper size, etc. in the whole stylesheet as a parameter.
xsl:attribute-set	Groups and defines the property, such as block and inline, for every object of XSL-FO to output.
xsl:template match="xxx"	Defines the template for transformation described to every element of an XML document ("xxx") . It is called by <xsl:apply-templates />
xsl:template name="yyy"	A subroutine of templates which is explicitly invoked by <xsl:call-template name="yyy"/>
xsl:key	Generates a key for an index. How to make an index is explained later in this document.

xsl:param and xsl:attribute-set are defined in param.xsl and attribute.xsl respectively, and are included in SD2 FO-DOC.XSL.

xsl:param and xsl:attribute-set are not necessarily required, but they have the following advantages:

- 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 value. -->
<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 name="keep-together.within-page">always</xsl:attribute>
</xsl:attribute-set>
```

From here, the stylesheet is explained along with this SD2 FO-DOC.XSL.

Page Layout specification

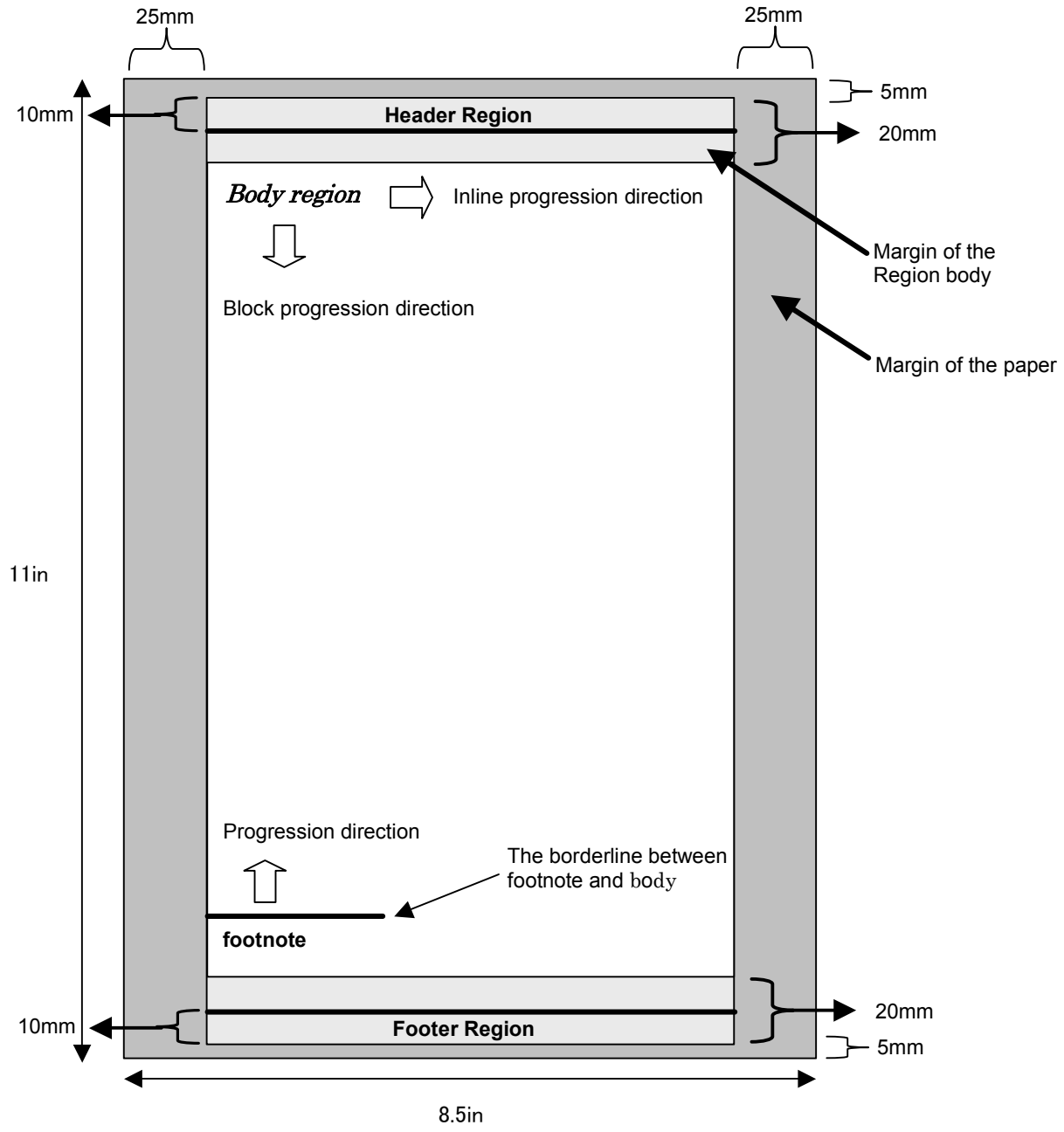
The page layout of SD2 FO-DOC.XSL has the following features:

- The page layout includes a cover, a table of contents, the text, and indexes.
- The page layout for indexing a cover, a table of contents places neither a page number nor a document name. Therefore, it does not have header/footer regions.
- The page layout for the text is different by right and left pages, and aligns the content of the footer region to the fore-edge side.
- The top page of the body text is made as the 1st page.
- Only the index page has two columns.

Therefore, 5 types of page layouts, a cover, a table of contents, the text (left), the text (right), and an index are required. The definition method of each page layout is described henceforth.

Page layout of Cover/Teble of contents

Page layout of Cover/Teble of contents is shown as follows:



The layout of Cover/Teble of contents

The page layout is defined as page master. Specifically, it's described as follows using the fo:simple-page-master element.

Definition of the page layout in the stylesheet

```
<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" />
```

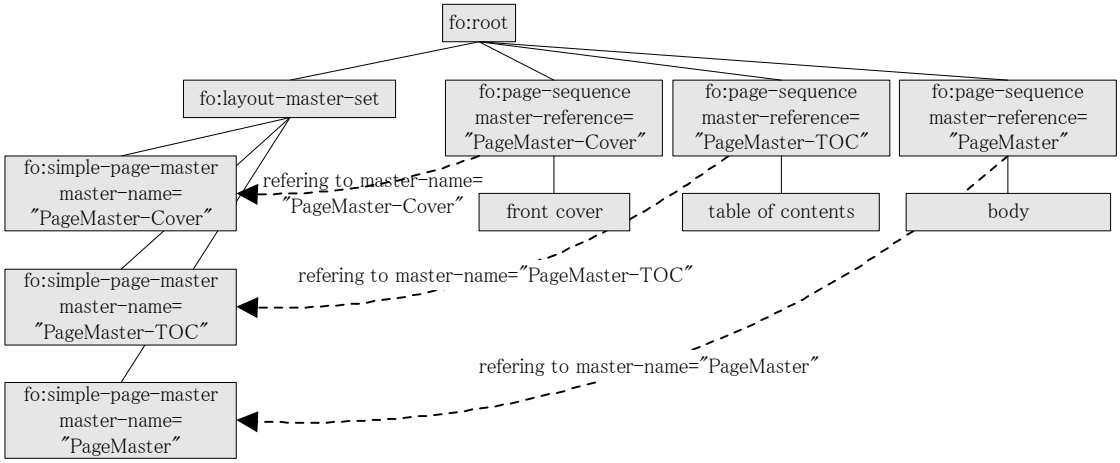
```
 </xsl:attribute>
 <fo:region-body  margin="0mm 0mm 0mm 0mm" />
</fo:simple-page-master>

<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>
```

Although the specified value is the same, the page master is prepared for each considering the possibility of any changes.

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

master-name	Usage	Refered Template
PageMaster-Cover	For a cover	<xsl:template match="doc/head">
PageMaster-TOC	for a table of contents	<xsl:template name="toc">

The next figure shows the XSL-FO tree structure in the site of where to define these page masters and where to refer to.



FO tree structure relating to page layout

Body - Change Page Layout on Right and Left pages

In this document, the page layout changes by right and left page. In XSL-FO, the layout change on right and left page can be made by grouping the even-page layout and odd-page layout, then changing by turns.

Create two fo:simple-page-master, one for left pages and another for right pages, and groups them by fo:page-sequence-master. fo:repeatable-page-master-alternatives is used for repeating two page layouts by turns. The odd-or-even property of fo:conditional-page-master-reference is used to specify the conditions on which the selection of the alternative is made.

It is described in the stylesheet as follows:

FO tree structure relating to page layout

```
 <fo:simple-page-master margin="10mm 00mm 10mm 00mm" master-name="PageMaster-Left">
  <xsl:attribute name="page-height">
   <xsl:value-of select="$paper-height-default" />
  </xsl:attribute>
  <xsl:attribute name="page-width">
   <xsl:value-of select="$paper-width-default" />
  </xsl:attribute>
  <fo:region-body margin="15mm 25mm 15mm 25mm" />
  <fo:region-before region-name="Left-header" extent="10mm" display-align="after" />
  <fo:region-after region-name="Left-footer" extent="10mm" display-align="before" />
  <fo:region-start region-name="Left-start" extent="20mm" />
  <fo:region-end region-name="Left-end" extent="20mm" />
 </fo:simple-page-master>

 <fo:simple-page-master margin="10mm 00mm 10mm 00mm" master-name="PageMaster-Right">
  <xsl:attribute name="page-height">
   <xsl:value-of select="$paper-height-default" />
  </xsl:attribute>
  <xsl:attribute name="page-width">
   <xsl:value-of select="$paper-width-default" />
  </xsl:attribute>
  <fo:region-body margin="15mm 25mm 15mm 25mm" />
  <fo:region-before region-name="Right-header" extent="10mm" display-align="after" />
  <fo:region-after region-name="Right-footer" extent="10mm" display-align="before" />
  <fo:region-start region-name="Right-start" extent="20mm" />
  <fo:region-end region-name="Right-end" extent="20mm" />
 </fo:simple-page-master>

 <fo:page-sequence-master master-name="PageMaster">
  <fo:repeatable-page-master-alternatives>
   <fo:conditional-page-master-reference master-reference="PageMaster-Left" odd-or-
even="even" />
   <fo:conditional-page-master-reference master-reference="PageMaster-Right" odd-or-
even="odd" />
  </fo:repeatable-page-master-alternatives>
 </fo:page-sequence-master>
```

Index □ Two-Column Layout

The page layout of an index is two-column layout. In XSL-FO, the number of column is specified to the column-count property of fo:region-body.In other word, although column layout can be specified per page, the number of columns cannot be changed in the middle of a page. However, the block object can be arranged as span -all layout by the span="all" property. The column gap can be specified by the colomn-gap property belonging to fo:region-body. These page layouts are described in the following stylesheet.

Definition of page layouts in the stylesheet

```
 <fo:simple-page-master margin="25mm 25mm 25mm 25mm" master-name="PageMaster-index">
  <xsl:attribute name="page-height">
   <xsl:value-of select="$paper-height-default" />
  </xsl:attribute>
  <xsl:attribute name="page-width">
   <xsl:value-of select="$paper-width-default" />
  </xsl:attribute>
  <fo:region-body margin="00mm 00mm 00mm 00mm" column-count="2" column-gap="20mm" />
 </fo:simple-page-master>
```

Output Control as a Whole

- 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 the body of document.
- 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 controlled 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. As required they are creaged in order of fo:layout-master-set outputs, a front cover, a table of contents and the body text. It is possible for the attributes of doc elements to control whether to make these or not. It is possible to control whether to output a cover/a table of contents or not by specifying the properties of the doc element or the external parameters. For example, a cover is not outputted by specifying <doc cover="false">. A table of contents is not outputted by specifying 'false' to the value of toc-make.

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>
```

Cover

- Output title, date and author to a cover page. which are children of the head element in the cover page. but not output the abstract.
- The width of the block that contains the title is 130mm, the height is 30mm, 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 sans-serif. 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 serif. 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 serif. When an image of logotype is specified to the author, place it preceding the author.

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

The layout specification of a title portion is arranged by the portion of name="cover.title" in xsl:attribute-set.

Layout specification of title, author, 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">"sans-serif"</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">30mm</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">"serif"</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">"serif"</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>
```

Followings shows the point which should be aware of:

- In order to layout the title, we use fo:block-container.
- The Property space-before ="25 mm" is specified to the fo:block-container. This title becomes the first block of the region body. However, space-before in the first block of this region body will be discarded by default, and a title will be arranged at the upper end of this region body. By considering as space-before.conditionality="retain", a space can be obtained compulsorily.

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. The example of an author name with a picture may be shown at the cover of this document.

Templates that transform the head elements

```
<xsl:template match="doc/head">
  <fo:page-sequence master-reference="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 start-indent="0mm">
    <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. xsl:use-attribute-sets calls a group of properties defined by the xsl:attribute-set element and applies them to fo:block-container that matches to each title, date and author. In each fo:block-container, each template is applied to each element of title, date, and author.

Table of Contents

- A table of contents is positioned next to the cover by feeding a page. The title is "Table of Contents". The background color is gray.
- A table of contents is created by collecting the title elements of of part, chapter, section, subsection, and subsubsection in an XML document.
- The contents of each line consist of the each title in the part, chapter, section, subsection, leaders (rows of dots) and 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.
- The internal link from each line of a table of contents to the title in the text is set for a PDF output.

Templates for Creating a table of contents

A 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-reference="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 a table of contents in the following order.

1. Create a new page-sequence. This page-sequence refer to fo:simple-page-master described in master-reference="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 attribute-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 parts, characters, sections, subsections, subsubsections of the whole document. Then, each node is sent to toc.line template that processes a line of the contents. Lines of a table of contents are alined in the order of appearance of the corresponded node in the XML document tree.

This template is called from the template that processes the doc elements. So, "current node" is the doc element node. xsl:for-each change this current node into each node group specified by the select attribute. Therefore, the 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.xsl:for-each

Templates for Creating Lines of TOC

The toc.line template creates a line of contents

toc.line templates that create 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>
```

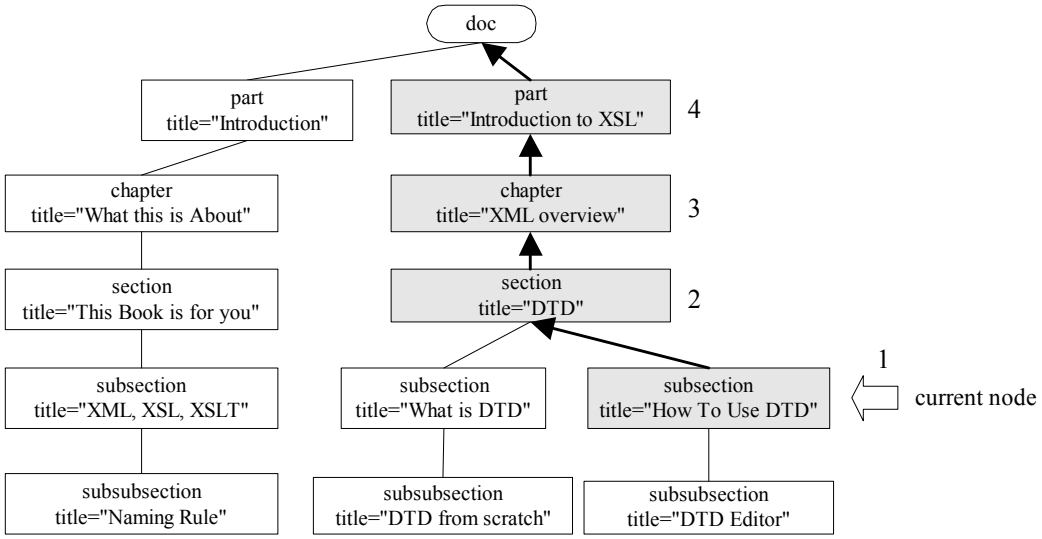
```
          <xsl:otherwise>1pt</xsl:otherwise>
        </xsl:choose>
      </xsl:attribute>
      <!-- font-size is processed in the same way-->
      <xsl:attribute name="font-size">
        <xsl:choose>
          <xsl:when test="$level=1">1em</xsl:when>
          <xsl:otherwise>0.9em</xsl:otherwise>
        </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. Formatter replaces it to the page number. --
>
      <fo:page-number-citation ref-id="{generate-id()}" />
    </fo:block>
  </xsl:if>
</xsl:template>
```

The toc template processes a table of contents in the following order.

1. Count the depth of the nest of the current node (nest level) from the root node, then 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 property of the doc element. If the level is not specified, the value 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 title, leader, and page number which is the real data of a line in a table of contents. Enclose the title of a table of contents with fo:basic-link and set the link from the title to the body text. Generated PDF is set as an internal link. (Details about fo:basic-link are explained in the 'Functions for creating PDF' section in this document.)

Counting the Nest Level

The nest level can be counted 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 which is a child of the doc elemnet 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 ancestor 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

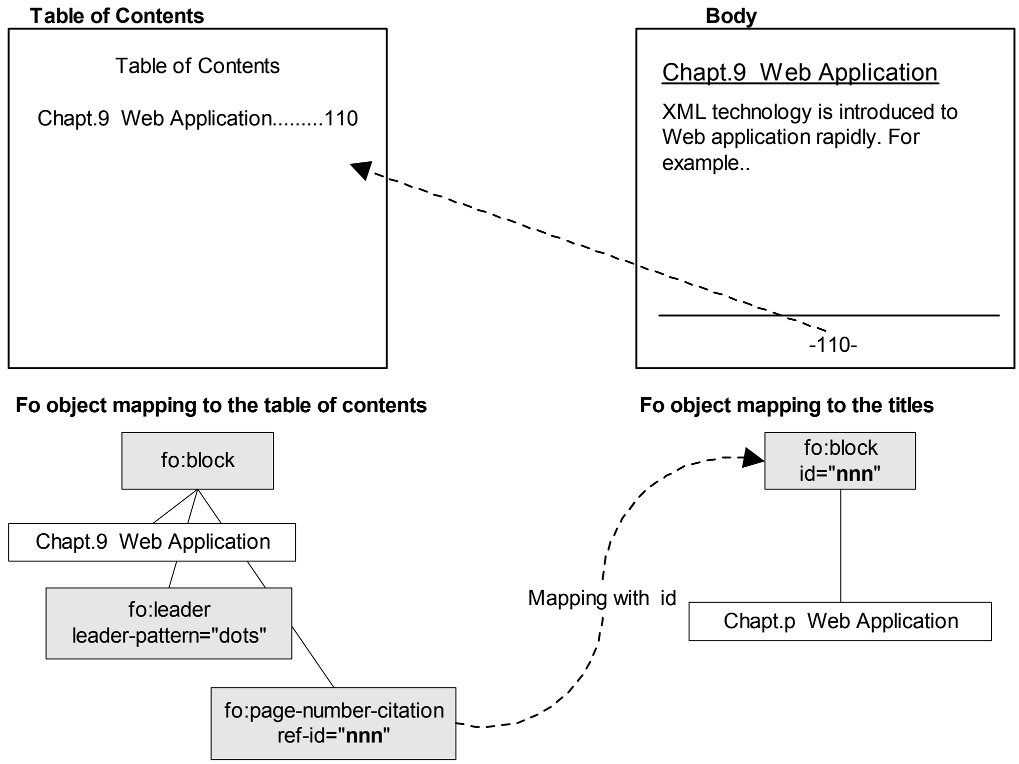
Setting properties according to the nest level

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.

Property	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

Getting 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 finishes formatting the XSL-FO instance.** In order to solve this problem, XSL-FO provides the function fo:page-number-citation. XSL 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. XSL Formatter finds the 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 is given. Therefore fo:block generated from the part, chapter, section, subsection elements must have the 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 the 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

The following pattern can also be specified.

```
<fo:leader leader-pattern="use-content">+</fo:leader>
```

Example of the generated contents

Shown below is an example of toc created by 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.

Body

- Process all the descendants or self of the body elements in the source XML document.
- Each page layout of the body consists of a page header, a page footer, and a body region. The contents of a page header and a page footer are arranged in a symmetrical position by odd-numbered and even-numbered pages.
- When a page has a footnote, the line which divides body region with footnote region is drawn.

Templates for Processing a Body

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

Templates that process the body element

```
<!-- process of body element -->
<xsl:template match="body">
<!-- start page number is 1 -->
    <fo:page-sequence master-reference="PageMaster" initial-page-number="1">
        <!-- left page header -->
        <fo:static-content flow-name="Left-header">
          <!-- snip-->
        </fo:static-content>
        <!-- right page header -->
        <fo:static-content flow-name="Right-header">
          <!-- snip-->
        </fo:static-content>
        <!-- left page footer -->
        <fo:static-content flow-name="Left-footer">
          <!-- snip -->
        </fo:static-content>
        <!-- right page footer -->
        <fo:static-content flow-name="Right-footer">
          <!-- snip -->
        </fo:static-content>

        <!-- leader for footnote-separator -->
        <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>

        <!-- body -->
        <fo:flow flow-name="xsl-region-body">
            <fo:block>
                <xsl:apply-templates />
            </fo:block>
        </fo:flow>
    </fo:page-sequence>
</xsl:template>
```

This template pocesses the body elements as shown below.

1. Generate a fo:page-sequence based on the new "PageMaster", the page layout right after the table of contents is changed.
2. Set header/footer regions based on the page layout. 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.

A page header and a page footer are described in fo:static-content. In the body page, in order to change the page layout on right and left pages, four fo:static-content should be prepared which are used for the right and left pages of the footer, and for the right and left pages of a header. Moreover, the border between the text and a footnote is created using also fo:static-content.

Four of fo:static-content are mapped to the region of a page as follows. The layout for right and left pages are defined in fo:simple-page-master of the body, which is prepared by the "Body － Change Page Layout on Right and Left pages " section in this document, then the header region and footer regions of each page are named, respectively. On the other hand, flow-name is specified to fo:static-content and the contents of fo:static-content are poured into the region whose name maps to theregion-name .

Page	Name of region	Name of static-content
Right page heade	fo:region-before region-name="Right-header"	fo:static-content flow-name="Right-header"
Right page footer	fo:region-after region-name="Right-footer"	fo:static-content flow-name="Right-footer"
Left page header	fo:region-before region-name="Left-header"	fo:static-content flow-name="Left-header"
Left page footer	fo:region-after region-name="Left-footer"	fo:static-content flow-name="Left-footer"

The border between a footnote and the text is created by fo:static-content with flow-name called xsl-footnote-separator. The fo:leader object is used for drawing a line. One third of the width of the body region is secured as a solid line.

The content of the body is outputted as a child of fo:flow.

Page Number Setting

It is possible to set up the initial value of a page number using the initial-age-number property belonging to fo:page-sequence. In SD2 FO-DOC.XSL, initial-page-number="1" is set to fo:page-sequence of the body, 1 page is made to begin from the body in SD2 FO-DOC.XSL.

Page Footer / Page Header

- Arrange a page number and a section title of the page in the page footer.
- Output a document title in a page header. Furthermore, a thumb index with number is arranged according to the appearance of the top-level element among part, chapter, or section in a document.

Page Footer Output

The contents of a page footer are outputted to fo:static-content. The contents are the same although the arrangement differs on a right and left page.

- Page number
- Section title of the page (Running footer)

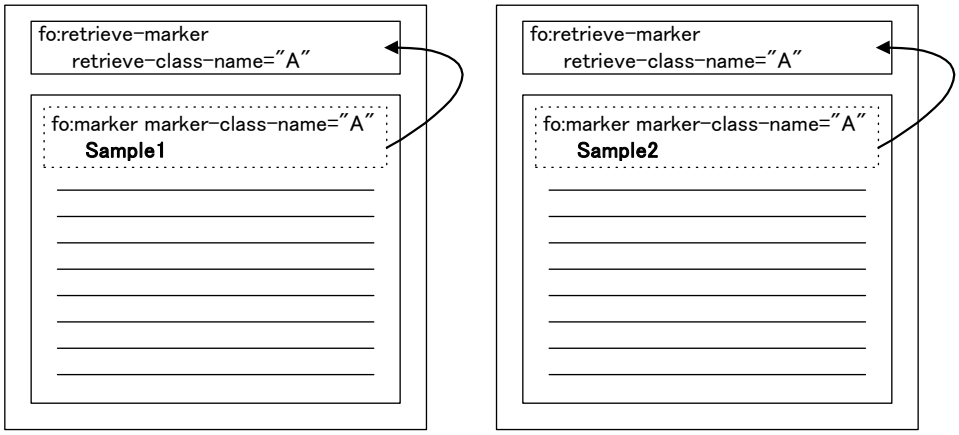
Page number

fo:page-number object is used for expressing a page number. fo:page-number object generates the special inline area, and XSL Formatter replaces it by the page number. In order to arrange a page number to a fore-edge side, text-align="outside" is specified to the propery of fo:block.

```
<!-- Place a page number to a footer region -->
    <fo:block font-size="9pt" text-align="outside">
        <fo:inline font-size="17pt">
            <fo:page-number />
            <xsl:text> | </xsl:text>
        </fo:inline>
    (Output a running footer (eplained later))
    </fo:block>
```

Running Footer

The title of the section in a document is outputted to a page footer. Since the title of section changes for every paragraph, it serves as a running footer. For this reason, fo:marker and fo:retrieve-marker are used. fo:marker is created to the title element in the text, and fo:retrieve-marker is put on fo:static-content of a page footer. XSL Formatter replaces the portion of fo:retrieve-marker by the corresponding contents while formatting.



Specify the class name of fo:marker you want to replace with the retrieve-class-name property belonging to fo:retrieve-marker. retrieve-boundary specifies a range of appliance. retrieve-position specifies which fo:marker in the page should be sellected. (What appeared first or what appeared last, for example)

```
<!-- Set retrieve-marker to output a section name to the footer. -->
<fo:retrieve-marker retrieve-boundary="page-sequence" retrieve-position="first-
starting-within-page" retrieve-class-name="section-title" />
```

Generates fo:marker against the title element in the text. It is described as follows in the template which processes "part | chapter | section | subsection | subsubsection | appendix."

```
<xsl:if test="local-name() = 'section'">
    <xsl:element name="fo:marker">
        <xsl:attribute name="marker-class-name">section-title</xsl:attribute>
        <xsl:value-of select="title" />
    </xsl:element>
</xsl:if>
```

Hereby, whenever a section appears in a document, fo:marker is generated as follows.

```
<fo:flow flow-name="xsl-region-body">
    <fo:marker marker-class-name="section-title">Preface</fo:marker>
    <!-- Contens of the section -->
    ....................................
    ....................................
    ....................................
    ....................................
    <fo:marker marker-class-name="section-title">Step for XSL-FO Transformation</
fo:marker>
    <!-- Contens of the section -->
    ....................................
    ....................................
    ....................................
    ....................................
    <fo:marker marker-class-name="section-title">SimpleDoc Organization</fo:marker>
    <!-- Contens of the section -->
    ....................................
    ....................................
```

Page Header Output

The contents of a page header are the following two.

- Title
- Thumb index

Title

The processing for outputting a title to a page header is only to output the title element to fo:block as follows.

```
<fo:block font-size="7pt" text-align="center" border-after-width="thin" border-after-
style="solid">
    <xsl:value-of select="/doc/head/title" />
</fo:block>
```

Thumb index

It's also possible to output a thumb index to a page using fo:marker and fo:retrieve-marker like the section title of a page footer. In SD2 FO-DOC.XSL stylesheet, 15 types of class names (thumb1, thumb2, ..., thumb14, and thumb0) are prepared and set to the objects which appear in a document (the top level object among part/chapter/section) in order. Each fo:retrieve-marker is set to fo:static-content which is a page header using a table cell. fo:retrieve-marker is not replaced if fo:marker which corresponds in a page does not exist. Thereby, the thumb index can be seen as moving according to the change of a section.

A table model for creating the page header in the stylesheet is as follows:

Creating a table for thumb indexes

```
<!-- a table with absolute-position -->
<fo:block-container absolute-position="fixed" top="0mm" left="20mm" height="15mm">
    <fo:table>
        <fo:table-column column-width="12mm" number-columns-repeated="15" />
        <fo:table-body>
            <fo:table-row>
                <fo:table-cell>
                   <fo:block font-size="24pt" text-align-last="center" color="white"
background-color="black" display-align="center">
                     <!-- It is replaced if a marker with the class of thumb1 exists.
-->
                        <fo:retrieve-marker xsl:use-attribute-sets="thumb-class"
retrieve-class-name="thumb1" />
                    </fo:block>
                </fo:table-cell>
                <!-- Create the rest 14 cells as well. (the last is retrieve-class-
name="thumb0".) -->
            </fo:table-row>
        </fo:table-body>
    </fo:table>
</fo:block-container>
```

In processing "part | chapter | section | --" in the text, the generation of fo:marker is shown as follows. The element name which changes a thumb index is set to Variable thumb ($thumb). Since it is the template which is common and processes two or more elements, you have to judge a current node. Using local-name(), the element name of a current node is investigated.

Creating fo:marker for thumb indexes

```
<xsl:template match="part | chapter | section | subsection | subsubsection |
appendix">
    <fo:block>
        <xsl:choose>
            <xsl:when test="(local-name() = 'part') and ($thumb = 'part')">
                <xsl:element name="fo:marker">
                    <xsl:variable name="num"><xsl:number format="1" /></xsl:variable>
                    <xsl:attribute name="marker-class-name">thumb<xsl:value-of
select="$num mod 15" /></xsl:attribute>
                    <xsl:number format="1" />
                </xsl:element>
            </xsl:when>
            <xsl:when test="(local-name() = 'chapter') and ($thumb = 'chapter')">
                <xsl:element name="fo:marker">
                    <xsl:variable name="num"><xsl:number format="1" /></xsl:variable>
                    <xsl:attribute name="marker-class-name">thumb<xsl:value-of
select="$num mod 15" /></xsl:attribute>
                    <xsl:number format="1" />
                </xsl:element>
            </xsl:when>
            <xsl:when test="(local-name() = 'section') and ($thumb = 'section')">
                <xsl:element name="fo:marker">
                    <xsl:variable name="num"><xsl:number format="1" /></xsl:variable>
                    <xsl:attribute name="marker-class-name">thumb<xsl:value-of
select="$num mod 15" /></xsl:attribute>
                    <xsl:number format="1" />
                </xsl:element>
            </xsl:when>
        </xsl:choose>
        <!-- Omit the rest. -->
```

As for the class name (marker-class-name), 'thumb' is set asa fixed character sequence, then a number is given in order after that. In order to begin from 1 and to repeat per 15 pieces, the element obtains first what position it is as compared with the same element in a document using xsl:number. Next, the number given to the class name is realized by using the number of the remainder after deviding the position figure by 15.

Head

- 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 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, sans-serif, Bold space-after: 14pt, bottom border: solid 2pt, break condition: break-before="page"
2	h2	Font : size 16pt, sans-serif, bold space-before: 19pt, space-after: 5pt, keep-with-next. within-page: always
3	h3	font: size 13pt, sans-serif, bold space-before: 14pt, space-after: 5pt, keep-with-next. within-page: always
4	h4	font: size 12pt, sans-serif, bold space-before: 5pt, space-after: 5pt, keep-with-next. within-page: always
5	h5	font: size 10pt, sans-serif, 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">"sans-serif"</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="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">"sans-serif"</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">"sans-serif"</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">"sans-serif"</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">"sans-serif"</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, it is kept at the top of the following page and is kept with the next block naturally.

Templates for Processing Heads

Templates that process the heads are shown below. Heads are processed in one template intensively because the styles are selected according to the nest level.

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

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

Although the template which processes 2nd part/title - subsubsection/title is "empty processing" (nothing is outputted), this is an usual practice for not outputting a title doubly.

The title line is outputted by the title.out template as shown proeviously. However the title character sequence will be outputted again because the title element appears while xsl:apply-templates processes the part/title - subsubsection/title elements. However, if the template which matches part/title - subsubsection/title is prepared and the contents are empty, the output of an excessive title character sequence can be controlled.

Example of a Generated Title

Shown below is an example of a generated title by taking previous steps. Note that the result of generate-id()function is stored as the value of the id property.

Generated titles

```
<fo:block font-size="24pt"
          font-family="&amp;quot;sans-serif&amp;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>
```

(2) Heads are referred from fo:page-number-citation of lines in a table-of-contents.

Processing Inline Elements

- The elements of b(bold), i(italic), em(emphasis), code(inline program code) are transformed into the fo:inline objects which specify the 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 property.
- 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.

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 property.

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 becomes [*italic*].

The inline element b becomes [**bold typeface**].

The inline element em also becomes [**bold typeface**].

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

a (anchor) Element

The anchor (a)element contains a Hypertext Reference property (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.

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*** / .'

Although the anchor was changed into the simple text here, in SD2 FO-DOC.XSL, thelink of PDF can be created by changing the anchor into fo:basic-link of XSL and formatted by XSL Formatter. Please refer to "Link Setting" of "Functions for creating PDF" in this document for details.

note Element

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

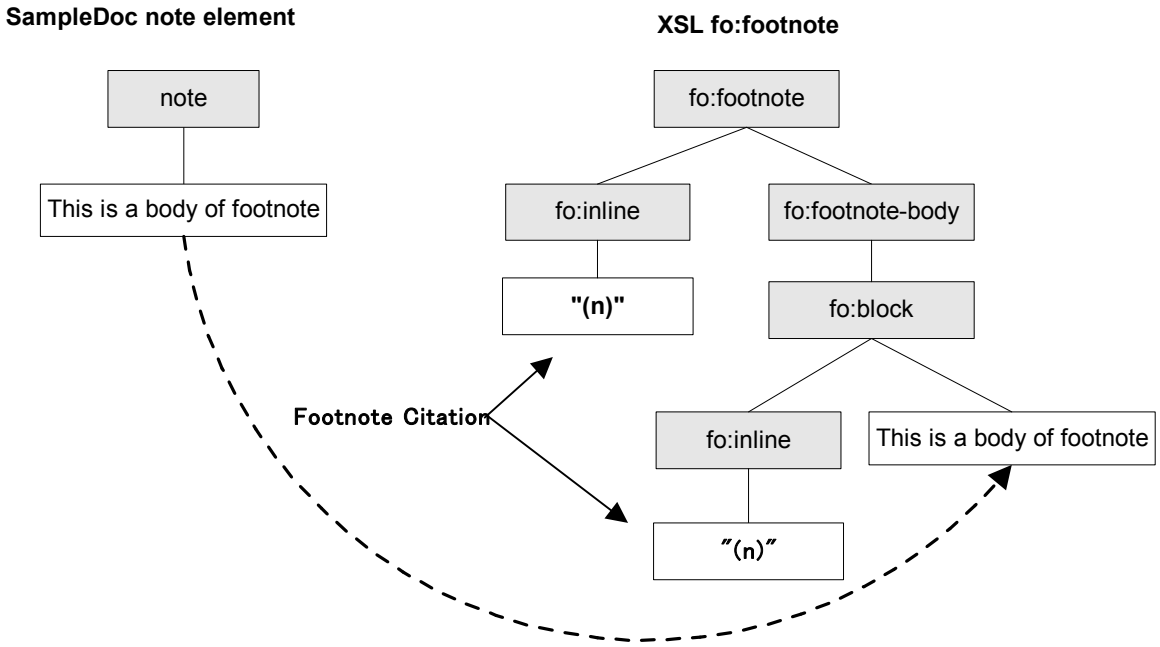
Template that processes note elements

```
<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 generates a footnote and a footnote citation.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. Please remark that you should specify all indent values of fo:footnote-body to zero in order to cancel the inheritance of indent value from the ancestor of fo:footnote.

Shown below is a typical fo:footnote object.



Typical example of the note element and fo:footnote

Generally, the same label 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 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 an XSL processing instruction. <xsl:number level="any" count="//note" format="(1)" /> searches all the descendant of the note elements under the root element in appearing order, finds out the same element as the current note element and 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 at the lower end of the body resion and separated from the text part.</note>" is expressed as:
"This is an example of footnote. Place a footnote here. (3)"

br Element

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

Templates that process the br element

```
<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 property 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 property set before the line break can be used."

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>
```

(3) This is a footnote text. It is placed at the lower end of the body resion and separated from the text part.

Processing Block Elements

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

- 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 caption, place it under 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 caption in front of the program code.
- Transform div(general block element) into fo:block.

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 beginning 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.

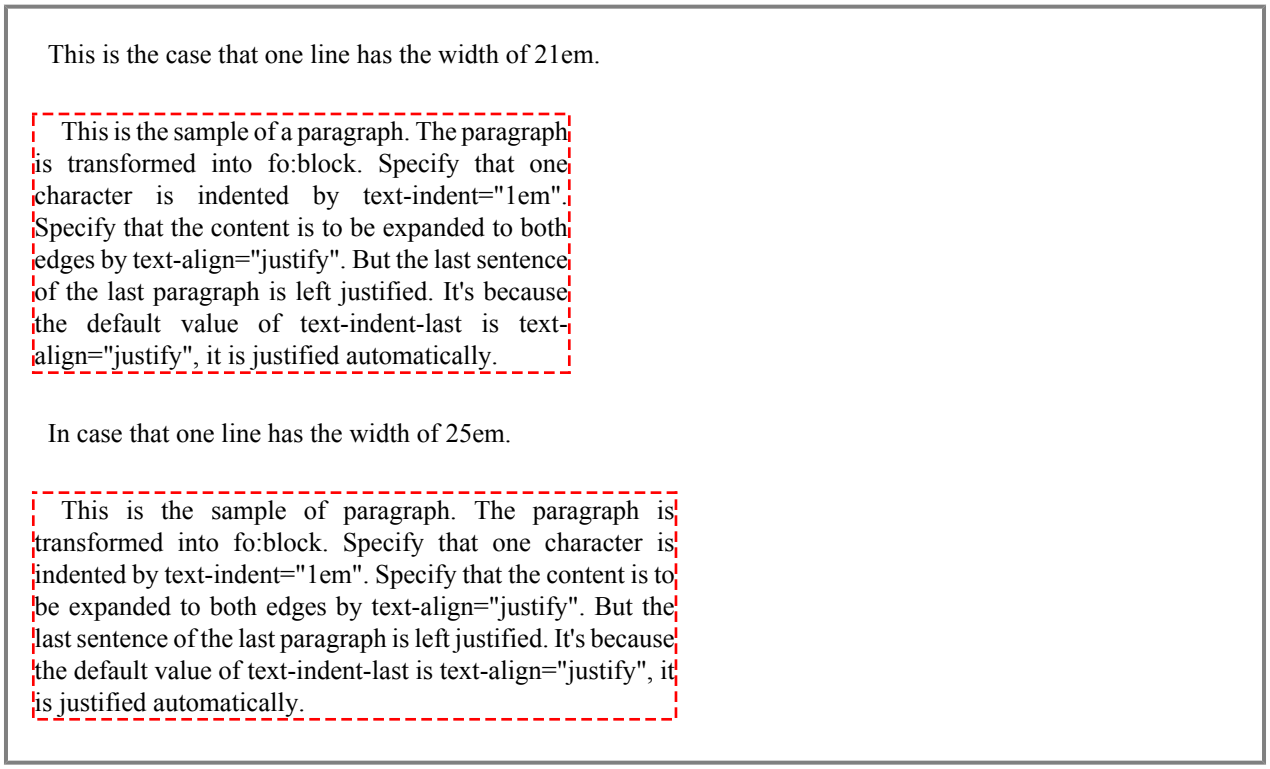


figure Element

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 property of the 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, TIFF, GIF, PNG, EPS, SVG, etc.

program Element

Program elements (Program code) are transformed into fo:block and display 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(). There are no templates to process text nodes (match="program/text()"). So they are processed by the built-in template in XSL in order to process only the 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 [(4)].
  3. white-space-collapse="false" : White spaces after processing by linefeed-treatment, space-treatment are not collapsed.
  4. wrap-option="no-wrap" : No line-wrapping will occur if the line overflows.

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

- Specify the value of wrap-option as wrap to override the default value of space-treatment. The line wraps if the line overflows (5).

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

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>
```

You may change this template as you desire. 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, you must specify following 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:

(5) wrap-option="wrap" is a default value. But it changes to wrap-option="no-wrap" by specifying white-space="pre". This property overides 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 s t y l e s h e e t limitation. But it is very **troublesome** to write FO directly.

Processing Table Elements

- 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 discriminate 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 padding, of which size is 0.3 x fontsize on the left, 0.2 x fontsize on the right, 2 point on the top and 2 point on the bottom.
- Process the properties related to the table format and make them reflected in the formatted result.
  1. layout propery, width property and rowheight property of the table element.
  2. number, property. width property of the col element
  3. height property of the tr element
  4. align property, valign property, colspan property, rowspan property defined for th, td elements.

Comparing SimpleDoc 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 SimpleDoc table with XSL table. First of all, the SimpleDoc table is as follows:

Element	Meaning	Definition
table	entire table	(title?, col*, thead?, tfoot?, tbody) Specify whether the table layout is formatted automatically or it is fixed, in the layout property. The entire width is specified in the width property.
col	column property	EMPTY The column width is specified in the width property, the column number is specified in the number property.
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 property.
th	table header cell	(a group of inline elements)* The number of the rows to be expanded across is specified in the colspan property, the number of the rows to be expanded down is specified in the rowspan. The align, valign propertys allows horizontal, vertical alignment to be set.
td	table data cell	Same as the definition of th

On the other hand, the object of the table of XSL-FO has the following composition.

Element	Meaning	Definition
fo:table-and-caption	entire table and caption	(table-caption?, table)
fo:table-caption	caption of the table	(%block;)(6)

Element	Meaning	Definition
fo:table	entire table (except caption cells).	(fo:table-column*, fo:table-header?, fo:table-footer?, fo:table-body+) table-layout : Specify automatically 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	table column	EMPTY column-number : The number of columns (7) column-width : The width of the column number-columns-repeated : The number of columns specifying the repetition of the table-column. number-columns-spanned : The number of columns spanned by table-cells.(8)
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 to be spanned.

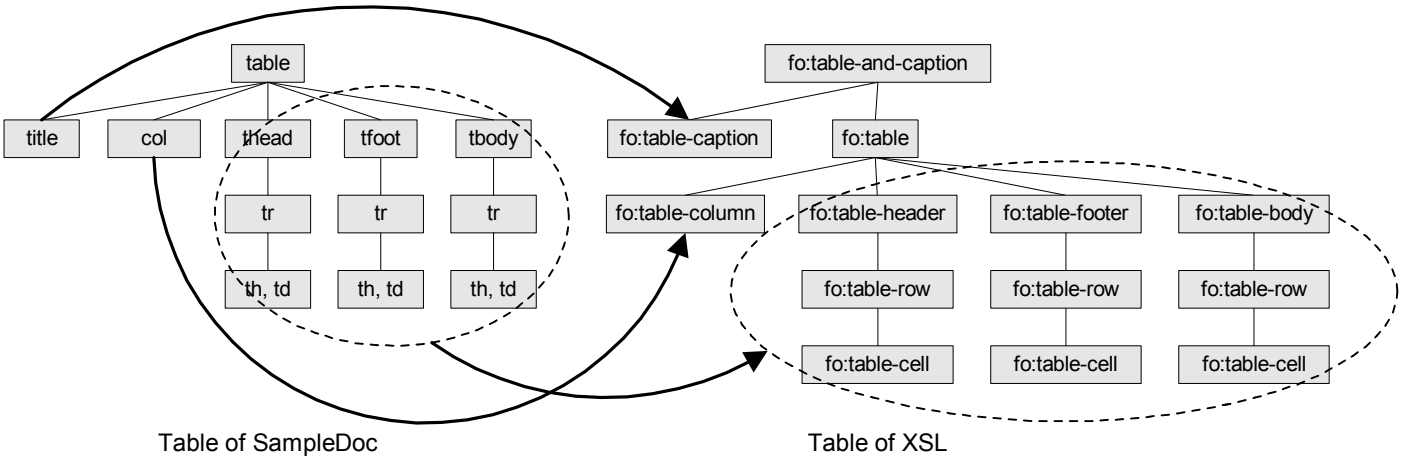


Table structure of SimpleDoc, XSL-FO

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

Templates that process tables

Templates that process tables are shown below.

(6) %block; is defined to contain block, block-container, table-and-caption, table, list-block in the XSL Specification. Generally, it is fo:block, fo:block-container.
(7) Use as a parameter of from-table-column() function.
(8) Refer to from-table-column() function.

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 name="padding-before">2pt</xsl:attribute>
  <xsl:attribute name="padding-after">2pt</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 name="padding-before">2pt</xsl:attribute>
  <xsl:attribute name="padding-after">2pt</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/title">
</xsl:template>

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

<xsl:template match="thead">
  <fo:table-header start-indent="0pt" end-indent="0pt" >
    <xsl:apply-templates />
  </fo:table-header>
</xsl:template>

<xsl:template match="tfoot">
  <fo:table-footer start-indent="0pt" end-indent="0pt" >
    <xsl:apply-templates />
  </fo:table-footer>
</xsl:template>

<xsl:template match="tbody">
  <fo:table-body start-indent="0pt" end-indent="0pt">
    <xsl:apply-templates />
  </fo:table-body>
</xsl:template>

<xsl:template match="tr">
  <xsl:element name="fo:table-row">
    <xsl:if test="@height">
        <xsl:attribute name="block-progression-dimension">
          <xsl:value-of select="@height"/>
        </xsl:attribute>
    </xsl:if>
    <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 SimpleDoc to that of XSL formatting object. Below shows the points.

1. The layout property defined for the table element is set for the layout property of fo:table to control the automatic table layout. The value of auto or fixed is valid.
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 (9).
4. The height property defined for the tr element is set for block-progression-dimension of the fo:table-row object.
5. The colspan, rowspan properties defined for the th, td elements are set to each number-columns-spanned, number-rows-spanned of fo:table-cell object. In addition, the align, valign propertieare set to text-align property, display-align property of fo:table-cell. In this way the text in the cell can be aligned.

(9) Furthermore, there is a way to specify an absolute value for a part of the columns, 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.

If nothing is specified, the table width will be divided by the number of columns and the width of the column will be given 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. 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 those 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.

When you specify the vertical alignment in the cell of the table, be sure of the following point. vertical-align can be specified in HTML, vertical-alignment can be specified in CSS2, however, the property which corresponds to this in XSL becomes display-align instead of vertical-align. valign also maps to display-align in the stylesheet shown here. In XSL, vertical-align is applied only to the element of the inline level element.

Processing List Elements

In SimpleDoc, three list elements with which the format of the label portion of an item differs, a list with (1) number (ordered list), a (2) numbers-less list (unordered list), and (3) definition type list [10] (definition list) are defined.

- The list element (ol, ul, and dl) of SimpleDoc is constructed in the format with which the label and the body part of the list are lined in horizontal mode.
- But, the definition type element (dl) also make it possible to format that the label and the body of the list are lined in vertical mode. [11]
- 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 SimpleDoc 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 neccessary to transform them to fo:list-block in XSL. Let us compare lists of SimpleDoc with those of XSL formatting objects.

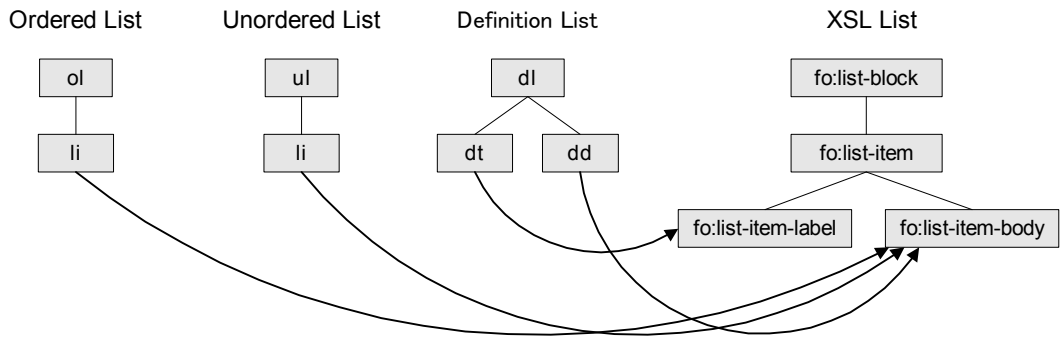
Element	Meaning	Definition
ol	ordered list	(li)*
ul	unordered list	(li)*
li	list item	(a group of inline elements)*
dl	definition list	(dt, dd)*
dt	definition term	(a group of inline elements)*
dd	definition details	(a group of inline elements)*

On the other hand, the list of XSL-FO has the following structure.

Element	Meaning	Definition
list-block	formatting object that formats 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	list item including list label and list body.	(list-item-label,list-item-body)
list-item-label	label of a list item.	(%block;)+
list-item-body	body of a list item	(%block;)+

Both correspondence relation is shown in a figure.

[10] definition type list is the itemized statements which specify a label character sequence for every item like a glossary.
[11] This is the same display format as the browser format of dt in HTML.



List of SimpleDoc and XSL

The followings are remarkable differences between two.

- As for ul and ol in HTML, the browser generates the label of a list automatically. In XSL-FO, you have to create it in a label region by using the stylesheet. For such occasions, the flexibility is high and the label format which is equal to the practical use for publication can be made.
- As for dl, you should just arrange the contents of dt to a label region by using the stylesheet.
- There is no function to calculate the width of the fo:list-item-label formatting object in XSL-FO. An appropriate value must be specified in the stylesheet.

Each template is explained 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>
```

Specifying the positions of label and body

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

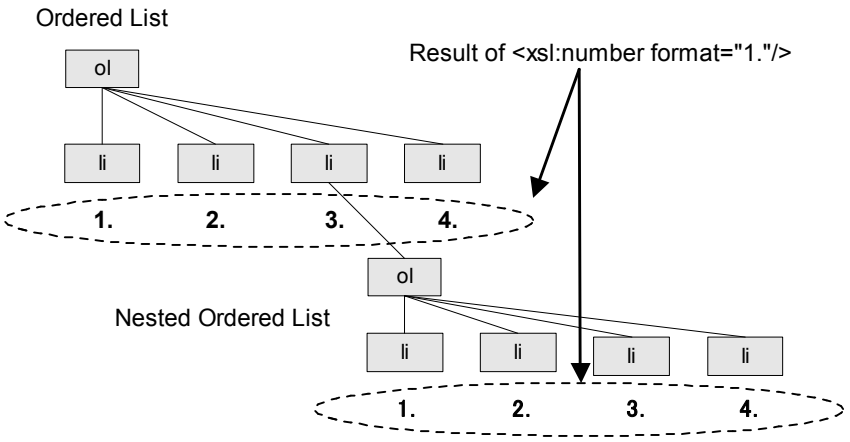
- provisional-distance-between-starts specifies the distance between the start of the list-item-label and the start of the list-item-body
- provisional-label-separation specifies the distance between the end of the list-item-label and the start of the list-item-body.

The template sets these values to the startdist, gap attributes of the ol element. But actually the list-item-label and list-item-body are specified as descendants of the fo:list-item. 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. (12) These two functions calculate the position by referring the values of provisional-distance-between-starts, provisional-distance-between-starts specified in fo:list-block. (13)

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. Numbers are processed correctly even if the lists are nested.



Ordered list labels processed by xsl:number

The label format is specified with the type property of the ol element. For example, the label (01), (02), (03)... will be generated if type="(01) is specified . The following formats can be used for a number portion.

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...
i	i, ii, iii, iv, v, vi, vii, viii, ix, x,...
I	I, II, III, IV, V, VI, VII, VIII, IX, X,...

The type property specifies UNICODE characters that represent zero, 1 and punctuation marks such as parentheses. As the label formatcan be specified in the XML source document, the document can be made flexibly.

(12) These are called 'Property Value Function' in the XSL Specification.

(13) label-end() = (the content-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)

Example of Ordered List

XML source data

```
<ol type="a.">
  <li>type of list
    <ol type="i)">
      <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. type of list
   i) unordered list
   ii) ordered list
   iii) definition list
b. list element
   1. row
   2. column
   3. cell
c. block element and inline element

Templates that Process the Unordered List

Templates for 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-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="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 "&#x2022;" -->
    <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>○</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>&#x2022;</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 property in the ul element.

Next template 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 property defined for the ul element.

Template that use an image as a character for label of line

```
<!-- The template that use an 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 occurs when transforming the definition list into fo:list-block formatting object.

- In the SimpleDoc 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 XSL stylesheest 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 name="text-align">justify</xsl:attribute>
</xsl:attribute-set>

<xsl:attribute-set name="dd.block" use-attribute-sets="dd.list">
  <xsl:attribute name="start-indent" >from-parent() + 4em</xsl:attribute>
  <xsl:attribute name="text-align">justify</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 as fo:list-block: the label (dt) and the body(dd) are lined in horizontal way.
- 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.

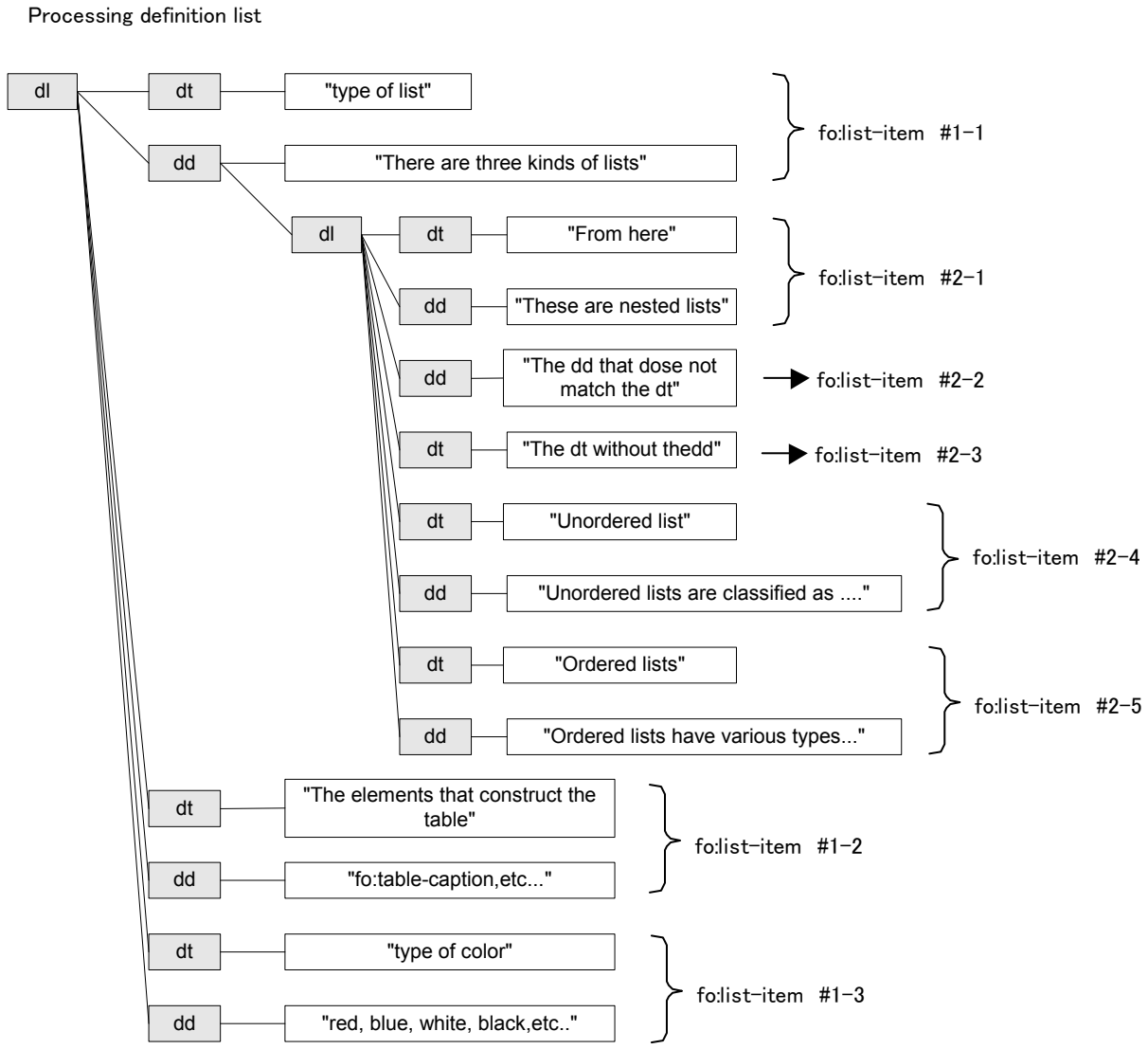
Example of a definition list

The following is a sample XML source data of definition list.

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 figure shows how 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.



Transforming the definition list into the list type.

The process.dl.list template calls itself recursively. In the general programming language, it is natural to assign a value to a variable and process using a variable. But in XSL, it is impossible to assign a value to a variable, but it is possible to initialize 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 appearing order.

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

type of list	There are three kinds of lists, which is unordered list, ordered list, and definition list.
From here	These are listed nests.
	The dd that the dd does not match dt.
The dt without the dd.	

	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. Also it is possible to specify colors by RGB() functions	

The following is the output of HTML type transformation.

type of list

There are three kinds of lists, which is unordered list, ordered list, and definition list.

From here

These are nested lists

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. Also it is possible to specify colors by RGB () functions.

Functions for creating PDF

With XSL Formatter, the following relates with PDF generation are available.

- Set the document information in the result PDF.
- Create bookmars in the result PDF.
- Set up the link to a PDF document (from a table of contents to the body text, for example).
- Set up the link to the external destination .

Although some of the functions described here in creating PDF are not defined in the specification of XSL-FO V1.0, it's possible by using Antenna House extension specification (Source Information [4]) .

PDF document information

Although the document information cannot be set up by XSL-FO V1.0 specification in case a formatted result is outputted to PDF, it's possible to set up the document information by Antenna House extension specification. The definition is given using axf:document-info. If axf:document-info is directly under fo:root, and it exists before fo:page-sequence appears, it will be reflected. In the style sheet, title, subtitle, and author are set up from "/doc/head" of the SimpleDoc as PDF document information as follows.

Outputting PDF document information

```
<xsl:template match="doc">
    <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
        <axf:document-info name="title" value="{/doc/head/title}" />
        <axf:document-info name="subject" value="{/doc/head/subtitle}" />
        <axf:document-info name="author" value="{/doc/head/author}" />
```

Creating a bookmark

PDF bookmarks use Antenna House extension specification (Source Information [4]) similarly. In this case, it becomes possible by specifying axf:outline-level and axf:outline-title to be fo:block to output to a bookmark. When there is no axf:outline-title, the contents of fo:block serves as a bookmark automatically. When processing the title of part | chapter | section | subsection | subsubsection | appendix, axf:outline-level is set up and the bookmark is made to output in SD2 FO-DOC.XSL. The template is <xsl:template name="title.out"> (sub template for outputting a title).

```
<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
         |ancestor-or-self::appendix )" />
        <xsl:choose>
            <xsl:when test="$level=1">
               <fo:block xsl:use-attribute-sets="h1" id="{generate-id()}" axf:outline-
level="{$level}">
                    <xsl:call-template name="title.out.sub" />
                    <xsl:value-of select="title" />
                </fo:block>
            </xsl:when>
```

```
            <xsl:when test="$level=2">
               <fo:block xsl:use-attribute-sets="h2" id="{generate-id()}" axf:outline-
level="{$level}">
                    <xsl:call-template name="title.out.sub" />
                    <xsl:value-of select="title" />
                </fo:block>
            </xsl:when>
            <xsl:when test="$level=3">
               <fo:block xsl:use-attribute-sets="h3" id="{generate-id()}" axf:outline-
level="{$level}">
                    <xsl:call-template name="title.out.sub" />
                    <xsl:value-of select="title" />
                </fo:block>
            </xsl:when>
            <xsl:when test="$level=4">
               <fo:block xsl:use-attribute-sets="h4" id="{generate-id()}" axf:outline-
level="{$level}">
                    <xsl:call-template name="title.out.sub" />
                    <xsl:value-of select="title" />
                </fo:block>
            </xsl:when>
            <xsl:when test="$level=5">
               <fo:block xsl:use-attribute-sets="h5" id="{generate-id()}" axf:outline-
level="{$level}">
                    <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>
```

Link setting

fo:basic-link is used to setup links . The reference destination id is specified using the internal-destination property to create a link into the same document. At this time, the XSL-FO object of the reference origin needs to have the id of the same value. In case of external links, the reference destination is specified with the external-destination property. In SimpleDoc, both internal/external links are expressed using the a element. The reference destination is specified by the href property and when the reference destination starts with "#", an internal link is set up, otherwise an external link is set up.

```
<xsl:template match="a[@href]">
    <fo:basic-link>
        <xsl:if test="starts-with(@href,'#')">
            <xsl:attribute name="internal-destination">
                <xsl:value-of select="substring-after(@href,'#')" />
            </xsl:attribute>
            <fo:inline xsl:use-attribute-sets="a">
                <xsl:apply-templates />
            </fo:inline>
        </xsl:if>
        <xsl:if test="starts-with(@href,'#')=false">
            <xsl:attribute name="external-destination">
```

```
                <xsl:value-of select="@href" />
            </xsl:attribute>
            <fo:inline xsl:use-attribute-sets="a">
                <xsl:variable name="anchor-texts">
                    <xsl:apply-templates />
                </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>
            </fo:inline>
        </xsl:if>
    </fo:basic-link>
</xsl:template>
```

The element with the name element as a reference destination is processed as an inline object which has id as follows.

```
<xsl:template match="a[@name]">
    <fo:inline id="{@name}">
        <xsl:apply-templates />
    </fo:inline>
</xsl:template>
```

Moreover, it is also possible to set a link to the portion where it corresponds in a document from a table of contents or an index. In this case, id used for reference creates id automatically using the generate-id() function. When the generate-id() function is called, XSLT processor will create a suitable character sequence like "IDXP83DL" corresponding to a current node for id.

In the case of a table of contents, the portion which outputs an item by the toc.line template is described as follows.

```
<fo:basic-link internal-destination="{generate-id()}">
    <xsl:value-of select="title" />
</fo:basic-link>
```

Set id using generate-id() also when outputting a title in the text used as the reference destination. The value of id generated by the generate-id() function is always the same among the same elements, and surely different against different elements.

Reference to Appendix

- Put data numbers automatically to the appendix list at the end of the book.
- Create the corresponding data numbers automatically for the reference to the reference data in the text.

Create an appendix list in the end of the book, and make it possible to refer to the appendix at the end of the book by the data numbers in the text. Even if it generates these data numbers automatically and revises the appendix list, it is made not to need to update the reference numbers in the text while editing.

The whole appendix list is expressed with the bib element, and each data is made into the contents of the li element in SimpleDoc.dtd. The ID number is given to the li element. Data numbers are automatically put at the appendix list. The stylesheet for this is as follows.

```
<xsl:template match="bib">
    <fo:list-block>
        <xsl:apply-templates select="li" />
    </fo:list-block>
</xsl:template>

<xsl:template match="bib/li">
    <fo:list-item end-indent="label-end()" id="{@id}">
        <fo:list-item-label>
            <fo:block>
                <xsl:text>[</xsl:text>
                <xsl:value-of select="position()" />
                <xsl:text>]</xsl:text>
            </fo:block>
        </fo:list-item-label>
        <fo:list-item-body start-indent="body-start()">
            <fo:block>
                <xsl:value-of select="." />
            </fo:block>
        </fo:list-item-body>
    </fo:list-item>
</xsl:template>
```

The appendix is referred by the ref element, ref-id="*xxx* "in the text. (*xxx* is the id value of the li element of the reference destination.) The stylesheet which generates this reference label is as follows.

```
<xsl:template match="ref">
    <xsl:variable name="target">
        <xsl:call-template name="get-position">
            <xsl:with-param name="id-value" select="@ref-id" />
        </xsl:call-template>
    </xsl:variable>
    <fo:basic-link internal-destination="{@ref-id}">
        <fo:inline>
            <xsl:text>(Source Information [</xsl:text>
            <xsl:value-of select="$target" />
            <xsl:text>])</xsl:text>
        </fo:inline>
    </fo:basic-link>
</xsl:template>
```

Since the reference destination of the ref element shows the turn of the li element in the bib element, then the position () function calculates what number the li element comes whose value of ref-id matches in the bib element.

Index

- Create indexes at the end of the book.

Here in this section, the creation method of an index is explained sequentially. The templates for indexes are put together in index.xsl. For more details, see also the contents of index.xsl.

The index element is defined by SimpleDoc for the items put on an index.

```
<xsl:key name="index-key" match="index" use="substring(@key,1,1)" />
    <xsl:key name="index-value" match="index" use="." />
```

Creating keys

Declare the key with a name using xsl:key in order to group the index elements by using the beginning of the character.

```
<xsl:key name="index-key" match="index" use="substring(@key,1,1)" />
    <xsl:key name="index-value" match="index" use="." />
```

Since xsl:key must be a top-level element within a stylesheet, it is described at the head of the SD2 FO-DOC.XSL stylesheet. The first xsl:key declares the head-one character of the key property to the key named index-key as a value. Like an English document, if it's possible to group the index elements by their contents, it's accomplished by the method using use="substring(1 ., 1) ", without using the key property. For the second xsl:key, the content of the index element is declared as a value to the key named index-value. This is used to sort and output keys in the grouped node set with the head-one character.

Creating an index page

Creation of an index is started from the index.create template in the index.xsl stylesheet. Like other pages, fo:page-sequence and fo:flow are generated in the beginning, and next, a title is outputted at the head of the index page.

Next, the index elements for numbers and alphabets are processed.

```
<xsl:template name="index.create">
    <fo:page-sequence master-reference="PageMaster-index">
        <!-- Set a document name in the header region. -->
        <fo:static-content flow-name="xsl-region-before">
            <fo:block font-size="7pt" text-align="center" border-after-width="thin"
border-after-style="solid">
                <xsl:value-of select="/doc/head/title" />
            </fo:block>
        </fo:static-content>
        <fo:flow flow-name="xsl-region-body">
            <fo:block>
                <!-- The title is spaned all.-->
                <fo:block span="all" line-height="15mm" background-color="#9bd49d"
text-align="center" font-size="20pt" space-after="10mm" axf:outline-level="1"
id="index-page">
                    INDEX
                </fo:block>
                <fo:block span="all">
```

```
                     
                </fo:block>
                <fo:block xsl:use-attribute-sets="index">
                    <!-- Numbers and alphabets. -->
                    <xsl:call-template name="index.create.mainALPHA" />
                   <xsl:if test="(/doc/@lang = 'ja') or (/doc/@lang = '') or not(/doc/
@lang)">
                        <!-- Japanese Kana -->
                        <xsl:call-template name="index.create.mainKANA" />
                    </xsl:if>
                </fo:block>
            </fo:block>
        </fo:flow>
    </fo:page-sequence>
</xsl:template>
```

Grouping and taking out the index elements

"index.create.mainALPHA" is the template which generates the index of alphabets. The following shows the examle using "index.create.mainALPHA"

What is considered to be difficult in creating an index is to group the target nodes and take them out. In order to group them, xsl:key which is mentioned above and the data in the head of index.xsl are used.

```
<!DOCTYPE DOCUMENT [
<!ENTITY ALPHA "'@1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ'">
```

Each alphabet character is defined as an entity and it is used for grouping. Characters are taken out one by one from the entity, and then the node set which has the character as a key is taken out.

```
<xsl:template name="index.create.mainALPHA">
    <xsl:param name="lettersALPHA" select="&ALPHA;" />
    <!-- (1) Process any one from 0 to 9, or from A to Z. -->
    <xsl:call-template name="index.create.section">
        <xsl:with-param name="letter" select="substring($lettersALPHA,1,1)" />
    </xsl:call-template>
    <!-- (2) Prepare for processing the following one from 0 to 9, or from A to Z. --
>
    <xsl:variable name="remainderALPHA" select="substring($lettersALPHA,2)" />
    <!-- (3) process the following one from 0 to 9, or from A to Z. -->
    <xsl:if test="$remainderALPHA">
        <xsl:call-template name="index.create.mainALPHA">
            <xsl:with-param name="lettersALPHA" select="$remainderALPHA" />
        </xsl:call-template>
    </xsl:if>
</xsl:template>
```

In the portion (1), one character used as a key is passed to the "index.create.section" template as a parameter, and a node set is processed in the "index.create.section" template. In (2), the following one character is taken out and all characters are processed by calling "$remainderALPHA" recursively by (3).

Node set output

The "index.create.section" template outputs a node set which has a character used as index-key.

```
<xsl:template name="index.create.section">
    <!--  Receive the character to be processed. -->
```

```
    <xsl:param name="letter" />
    <!-- Set the nodes who have the 'letter' value of the key to the terms variable
and sort them.-->
    <xsl:variable name="terms" select="key('index-key',$letter)" />
    <!-- Get the index which has the same head-one character as the parameter which
received. -->
    <xsl:if test="$terms">
        <!-- Output key characters. -->
        <fo:block font-weight="bold" text-align="center" space-before="1em">
            <!-- Output the received one character as is. -->
            <xsl:value-of select="$letter" />
        </fo:block>
        <!-- Make it a rule not to process the same text redundantly. -->
        <xsl:for-each select="$terms[not(.=preceding::index)]">
            <!-- Sort the text in key order. -->
            <xsl:sort select="@key" />
            <fo:block text-align-last="justify" axf:suppress-duplicate-page-
number="true">
                <!-- Output the text as is. -->
                <xsl:value-of select="." />
                <fo:leader leader-pattern="dots" leader-length.optimum="3em" />
                <!-- Get the same text. -->
                <xsl:variable name="terms2" select="key('index-value',.)" />
                <xsl:if test="$terms2">
                    <fo:inline>
                      <!-- Output all of those page numbers when the same text exists
in two or more places. -->
                        <xsl:apply-templates select="$terms2" mode="line" />
                    </fo:inline>
                </xsl:if>
            </fo:block>
        </xsl:for-each>
    </xsl:if>
</xsl:template>
```

By using the axf:suppress-duplicate-page-number property (Antenna House extension specification (Source Information [4])), It is possible to suppress the duplicate page numbers.

A

Antenna House 1,1,2,3,3,5,7,7,8

X

XSL-FO 2,10,11,11,13,13

Example result without using the axf:suppress-duplicate-page-number property

Example result using the axf:suppress-duplicate-page-number property

Miscellaneous

Using modes

The SD2 FO-DOC.XSL stylesheet includes the stylesheet called article.xsl. This stylesheet processes to format a document without a cover / table of contents / index, when class="article" is specified in the doc element of an XML document. mode is specified in all the templates as mode="article" and the processing is different from the template of the SD2 FO-DOC.XSL stylesheet. Thus, the processing of arbitrary elements can be divided by using the mode attribute for the template.

Appendix

Source Information

[1] Extensible Stylesheet Language (XSL) Version 1.0,
W3C Recommendation 15 October 2001,
http://www.w3.org/TR/2001/REC-xsl-20011015/

[2] XSL Transformations (XSLT) Version 1.0
W3C Recommendation 16 November 1999,
http://www.w3.org/TR/xslt

[3] PureSmartDoc
http://www.asahi-net.or.jp/~dp8t-asm/java/tools/

[4] Antenna House Extension Specification
http://www.antennahouse.com/xslfo/axf3-extension.htm

Update History

The original version of this tutorial was written in Japanese for XSL School, June 2001, Tokyo.

The English version of the tutorial was first released on October 23, 2001. It was based on Candidate Recommendation of XSL Specification.

Minor revised version was released on May 6th, 2002 in order to be conformant to Recommendation of XSL Specification Version 1.0.

This version, released on February, 2005, was made by adding the contents of indexes, functions for PDF creation, reference to appendix and much more substantial contents.

INDEX

R

S

T

U

V

X