Reference number of working document: **ISO/IEC JTC1 Nxxxx**

Date: 2001-10-03

Reference number of document: **ISO/IEC DTR xxxxx:2001**

Committee identification: ISO/IEC JTC1/SC34

Secretariat: XXXX

Information technology — Document Description and Processing Languages — DSSSL library for complex compositions

Titre — Titre — Partie n: Titre de la partie

Warning

This document is not an ISO International Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an International Standard.

Recipients of this document are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Document type: International standard
Document subtype: if applicable
Document stage: (20) Preparation
Document language: E

C:\data-dk2\1_standards\iso_tr_dsllib\011028\dtr_dsllib.doc Basic template BASICEN2 1999-02-12

Copyright notice

Contents

table of contents is an optional preliminary element, but is necessary if it makes the standard easier to consult. The table of contents shall be entitled «Contents» and shall list clauses and, if appropriate, subclauses with titles, annexes together with their status in parentheses, the bibliography, index(es), figures and tables. The order shall be as follows: clauses and subclauses with titles; annexes (including clauses and subclauses with titles if appropriate); the bibliography; index(es); figures; tables. All the elements listed shall be cited with their full titles. Terms in the «Terms and definitions» clause shall not be listed in the table of contents.

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Committee) together form a system for worldwide standardization as a whole. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organizations to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC1.

The main task of a technical committee is to prepare International Standards but in exceptional circumstances, the publication of a technical report of one of the following types may be proposed:

– type 1, when the necessary support within the technical committee cannot be obtained for the publication ofan International Standard, despite repeated efforts;

– type 2, when the subject is still under technical development requiring wider exposure;

– type 3, when a technical committee has collected data of a different kind from that which is normally published as an International Standard ("state of the art", for example).

This specification is a translation of a JIS/Technical Report "DSSSL library for complex compositions" (TR X 0010:2000) published by Japanese Standards Association (JSA) in September 01, 2000.

Introduction

This Technical Report is based on the activities taken by Application Standards Committee of JBMA (Japan Business Machine Makers' Association) and Electronic Publishing Committee of INSTAC/JSA for application technology of DSSSL (ISO/IEC 10179, Document Style Semantics and Specification Language).

The DSSSL library specified in this Technical Report encourages interchanging SGML (Standard Generalized Markup Language) or XML (Extensible Markup Language) documents with such a complicated style specification by DSSSL as multilingual composition including Japanese and English texts within a page, clause or paragraph.

Information technology—
Document Description and Processing Languages —
DSSSL library for complex compositions

1 Scope

This Technical Report provides a DSSSL library that can specify styles for the documents described by SGML (Standard Generalized Markup Language, ISO 8879) or XML (Extensible Markup Language, W3C REC-xml-980210). The library makes it feasible to describe DSSSL specification for those documents, without any particular knowledge of DSSSL or particular composition rules.

2 References

The following standards contain provisions that, through reference in this text, constitute provisions of this Technical Report. At the time of publication, the editions indicated are valid. All standards are subject to revision, and parties to agreements based on this Technical Report are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below. Members of IEC and ISO maintain registers of currently valid International Standards.

ISO/IEC 10179:1996, *Information technology — Processing languages — Document Style Semantics and Specification Language (DSSSL)*.

ISO 8879:1986, *Information technology — Text and office systems — Standard Generalized Markup Language (SGML)*.

ISO 8879:1986/Cor.2:1999, *Information technology — Text and office systems — Standard Generalized Markup Language (SGML) TECHNICAL CORRIGENDUM 2*.

ISO/IEC 9541-1:1991, *Information technology — Font information interchange — Part 1: Architecture*.

3 Definitions

For the purposes of this Technical Report, the following definitions apply.

3.1
bottom edge
The bottom side of a page.

3.2
cell
a basic unit in a table composition.

3.3
character size
The size of virtual body of a character.

3.4
clock numerals
A Roman numerical character designed for a clock face.

3.5
column
A separated composition area within a page.

3.6
column space
An interval between adjacent columns.

3.7
DSSSL library
A set of DSSSL specifications (see ISO/IEC 10179).

3.8
edge
One of the three sides of a page that are not bound. The three sides are top edge, bottom edge and front edge.

3.9
emphasizing mark
A symbol located at the side (right side in vertical composition or top side in horizontal composition) of a character to be emphasized.

3.10
facing page
A two-page (left page and right page) spread of a book.

3.11
group-ruby
A ruby scheme in which ruby characters are aligned at the side of a grouped object characters.

3.12
gutter
A binding portion of a book.

3.13
head line
Description of a book-title, chapter-title or section-title allocated on a margin areas of a page.

3.14
image area
An effective page area on which main texts are composed.

3.15
indentation
Line head shifting along the character progression direction in a line.

3.16
interline space (or line space)
An interval between adjacent lines.

3.17
line leading
Displacement between the corresponding points of adjacent lines.

3.18
mono-ruby
A ruby scheme in which ruby characters are aligned with respect to each object character.

3.19
number of characters on a line
The number of characters which are included and aligned in a line.

3.20
ordering character
A head character of each ordered object (list, clause, etc.), indicating the order of the object.

3.21
page number
The number indicating a page sequence.

3.22
pre/post-marking of ordering character(s)
Character(s) just before/after ordering character(s).

3.23
ruby
A (sequence of) small character(s) located at the side of object character(s), indicating the pronunciation, meaning, etc. of the object character(s).

3.24
single page
A separated page. It is employed specifically discriminating a page from a facing page.

3.25
solid matter (or solid)
A composition without space between adjacent characters or lines.

3.26
space
An interval between virtual bodies of adjacent characters or lines.

3.27
strike-through (or strike-out)
A line overwritten on characters indicating their deletion.

3.28
superscript/subscript (or superior/inferior)
Small characters allocated at the side (right-top or right-bottom respectively) of an object character.

3.29
table caption
A title (and other related description) of a table. It is sometimes referred to as table title or table heading.

3.30
top edge
The top side of a page.

3.31
two-character ruby
A ruby consisting of the character(s) whose virtual body size (measured in the character progression direction) is a half of the virtual body size of corresponding object character.

3.32
writing direction
The direction of a character string composing a line. Multilingual texts are composed with a vertical or horizontal direction. Those compositions are called vertical compositions or horizontal compositions respectively.

3.33
writing mode
A character progression direction on a line. [The direction to escapement point from position point on a glyph coordinate system (see ISO/IEC 9541-1).] Multilingual texts employ the writing modes of LEFT-TO-RIGHT, TOP-TO-BOTTOM or RIGHT-TO-LEFT.

4 Formatting objects and properties

This clause defines formatting objects and properties dealt with or related to this DSSSL Library.

4.1 Paper size

The following paper sizes are dealt with:

a) A-size: A6, A5, A4

b) B-size: B6, B5

The choice of "single page" or "facing page" can be specified. In choice of the facing page, double size of those paper sizes a), b) can be specified.

NOTE 1 In choice of facing page with A4 left and right pages, A3 is specified.

4.2 Paper placement

The choice of “vertical (portrait)” or “horizontal (landscape)” can be specified. Default specification is vertical.

4.3 Unit

The following units are dealt with:

a) mm

b) point (1point = 0.3514mm)

c) Q (1Q = 0.25mm)

The relationship between Q and points is clarified in Table 1

Table 1 — Relationship between Q and points

Q	points	Q	points
7	5	15	10.5
8	5.5	16	11
9	6	18	12
10	7	20	14
11	7.5	21	15
12	8	24	16
13	9	26	18.5
14	10	28	20

4.4 Basic composition style

Prior to the basic composition styles, there should be specified the following properties:

a) size of image area

b) margin

They have the following relationship with a paper size:

(paper size) - margin = (size of image area)

In a layout driven specification, the values of paper size and margin are given first. In a content driven specification, the values of paper size and image area are given first.

NOTE 2 In Japanese composition, for example, a content driven specification is usually employed.

Basic composition styles are defined by the following formatting properties:

a) vertical composition or horizontal composition

b) size of character

c) number of characters on a line

d) number of lines in a column

e) interline space

f) column space

g) alignment (left, right and center alignments, and justification. Default specification is a left alignment.)

The basic composition styles are specified by one of the following manners:

a) to specify all the values of required formatting properties for the specified image area

b) to specify a model of basic component style, i.e., a set of the property values (see 4.5)

4.5 Model of basic composition style

Typical sets of the property values for book compositions are defined.

4.5.1 Position of image area on a paper

A position of image area on a paper is defined by the following manners:

a) An image area is positioned at the center of the rectangle surrounded by the margins of top edge, bottom edge, front edge and gutter. It is the default positioning of image area.

b) An image area is positioned in accordance with the ratio of top edge, bottom edge, front edge and gutter.

c) An image area is positioned in accordance with each value of top edge, bottom edge, front edge and gutter.

4.5.2 Models of basic composition styles

Typical sets of the property values for book compositions are provided as models of basic composition styles for some combinations of (paper size, paper placement, unit). Seven models are shown in Table 2 through Table 8, where the table captions indicate the values of (paper size, paper placement, unit).

Table 2 — B6, Vertical, Point

Vertical/Horizontal composition	Size of character (points)	Number of characters on a line	Number of lines in a column	Line leading (points)	Number of columns	Column space (characters)
vertical	9	43	14	18	1	none
vertical	9	43	15	18	1	none
vertical	9	43	16	17	1	none
vertical	9	44	17	16	1	none
vertical	8	50	18	15	1	none
vertical	8	50	19	14	1	none
vertical	8	25	20	14	2	2
vertical	8	26	20	14	2	2
vertical	9	30	23	17	1	none
vertical	9	33	25	16	1	none
vertical	8	33	27	15	1	none
vertical	8	34	27	15	1	none

Table 3 — B5, Vertical, Point

Vertical/Horizontal composition	Size of character (points)	Number of characters on a line	Number of lines in a column	Line leading (points)	Number of columns	Column space (characters)
vertical	8	24	31	13	1	none
horizontal	9	43	32	18	1	none
horizontal	9	23	44	14	2	2
horizontal	9	22	41	15	2	2
horizontal	8	25	51	12	2	2

Table 4 — B5, Vertical, Q

Vertical/Horizontal composition	Size of character (Q)	Number of characters on a line	Number of lines in a column	Line leading (Q)	Number of columns	Column space (characters)
horizontal	13	42	31	26	1	none
horizontal	13	22	43	20	2	2
horizontal	13	21	39	22	2	2
horizontal	12	23	48	18	2	2

Table 5 — A6, Vertical, Point

Vertical/Horizontal composition	Size of character (points)	Number of characters on a line	Number of lines in a column	Line leading (points)	Number of columns	Column space (characters)
vertical	8	41	13	17	1	none
vertical	8	41	14	16	1	none
vertical	8	42	15	15	1	none
vertical	8	42	13	16	1	none
vertical	8	42	14	16	1	none
vertical	8	43	15	15	1	none
vertical	8	43	15	15	1	none
vertical	8	43	16	14	1	none
vertical	8	43	18	13	1	none
vertical	8	43	19	13	1	none

Table 6 — A5, Vertical, Point

Vertical/Horizontal composition	Size of character (points)	Number of characters on a line	Number of lines in a column	Line leading (points)	Number of columns	Column space (characters)
vertical	9	51	16	18	1	none
vertical	9	52	16	18	1	none
vertical	9	52	17	18	1	none
vertical	9	52	18	17	1	none
vertical	9	52	19	17	1	none
vertical	9	25	20	15	2	2
vertical	8	30	24	13	2	2
vertical	8	29	23	14	2	2
horizontal	9	35	26	18	1	none
horizontal	9	35	28	17	1	none
horizontal	9	35	30	16	1	none
horizontal	8	40	30	16	1	none
horizontal	8	38	33	14	1	none

Table 7 — A5, Vertical, Q

Vertical/Horizontal composition	Size of character (Q)	Number of characters on a line	Number of lines in a column	Line leading (Q)	Number of columns	Column space (characters)
horizontal	13	34	27	25	1	none
horizontal	13	34	29	23	1	none
horizontal	12	37	28	24	1	none
horizontal	12	36	31	21	1	none

Table 8 — A4, Vertical, Q

Vertical/Horizontal composition	Size of character (Q)	Number of characters on a line	Number of lines in a column	Line leading (Q)	Number of columns	Column space (characters)
horizontal	13	51	41	24	1	none
horizontal	14	48	39	25	1	none
horizontal	14	24	42	23	2	2
horizontal	14	16	42	23	3	2

4.6 Font

Fonts should be specified by their font family names and weights.

NOTE 3 Such variations as “horizontal condensed", "vertical condensed" or "slant" are not dealt with.

4.7 Unit of character size

Character sizes should be specified by using a unit of point or Q.

4.8 Headline

4.8.1 Number of headlines

As the number of headlines, no or one should be specified.

4.8.2 Position of headline

The following positions of headlines are dealt with:

a) at the front side on the top edge of odd page

b) at the front side on the top edges of odd and even pages

c) at the front side on the bottom edge of odd page

d) at the front side on the bottom edges of odd and even pages

e) at the center on the top edge (for horizontal composition)

The writing direction of headlines is horizontal. If a headline and page number are located on the same edge, the headline should be located inside and apart from the page number by a space of 1 em or 1.5 em.

A division of words in a headline should follow the rule in 4.15.

4.8.3 Contents of headline

When the contents of headlines on odd and even pages are different from each other, the headlines should be positioned in accordance with 4.8.2 b), d) or e).

If a headline shows a clause title (or a node of logical structure of the text), the headline should be located on a page opposite to the page where the clause begins. On the page where a clause begins, there should be no headline.

4.9 Page Number

The following positions and styles of page numbers are dealt with:

a) at the front side on the top edge, using Arabic numerals

b) at the front side on the bottom edge, using Arabic numerals

Characters for a page number should have the same size as characters for texts on the page or one size smaller than those. A page number should be located apart from the image area having a space of the same size as the characters for texts or one size larger than those.

4.10 Note

4.10.1 Type of note

The following five types of notes are dealt with:

4.10.1.1 Interlinenote

A interlinenote is composed within a line space and at the side (upper side in horizontal composition, or right side in vertical composition) of words or phrases to be noted. Character locations in a line and interlinenote in horizontal composition are illustrated in Figure 4.1.

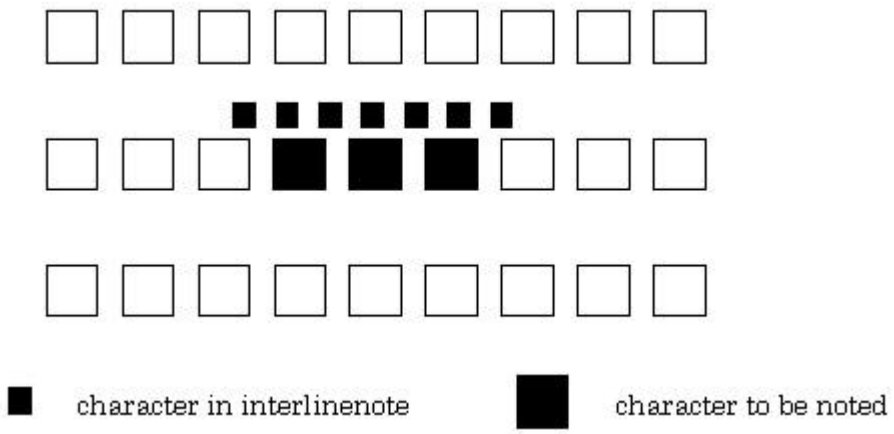

Figure 4.1 — Interlinenote in horizontal composition

4.10.1.2 Sidenote

A sidenote is used in horizontal composition and located on a note area which is reserved at the front side of a page as shown in Figure 4.2.

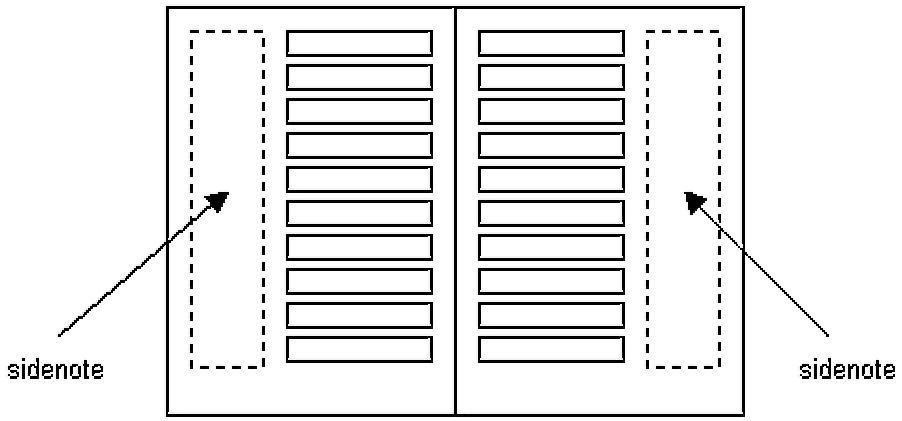

Figure 4.2 — Sidenote

4.10.1.3 Headnote

A headnote is used in vertical composition and located on a note area which is reserved at the top side of a page as shown in Figure 4.3.

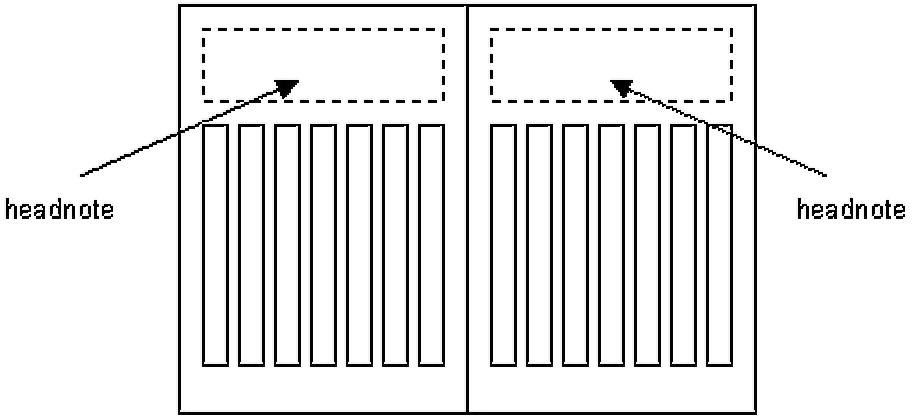

Figure 4.3 — Headnote

4.10.1.4 Footnote

A footnote is used in horizontal and vertical compositions and located on a note area which is reserved at the bottom side of a page as shown in Figure 4.4.

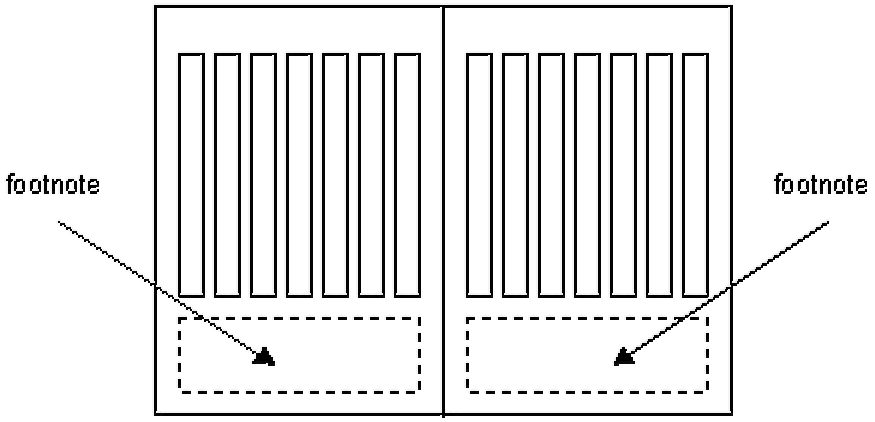

Figure 4.4 — Footnote in vertical composition

4.10.1.5 Endnote

A endnote is composed at the end of a paragraph, section, chapter or volume, or at the end of a book.

4.10.2 Reference Mark

A reference mark indicates the correspondence between a note and the text to be noted.

4.10.2.1 Character and style

For a reference mark, characters "*", "†"(dagger), "‡"(double dagger), "§"(section mark), "‖"(double bar), "#", Arabic numerals, Kanji numerals and Latin alphabet are employed. Some examples of reference marks using those characters are shown below:

Example 1	*	**	***
Example 2	*1	*2	*3
Example 3	1)	2)	3)
Example 4	(a)	(b)	(c)

Default reference marks are "1), 2), …".

4.10.2.2 Position

A reference mark within texts should be allocated to the following positions:

a) at the beginning of the words or phrases to be noted, with an offset to interline

b) at the end of the words and phrases to be noted, with/without an offset to interline.

4.10.2.3 Character size

A character size of reference marks should be one size smaller than the texts to be noted.

4.11 Inlinenote

An inlinenote consists of parenthesized two-line texts within a line as shown in Figure 4.5.

The composition of a inlinenote is based on that of inline (see 4.24). The parentheses have the same size as characters of the text outside the inlinenote. The default character size for inlinenote texts is a half of the character size for the texts outside the inlinenote. The default line space for inlinenote is 0.

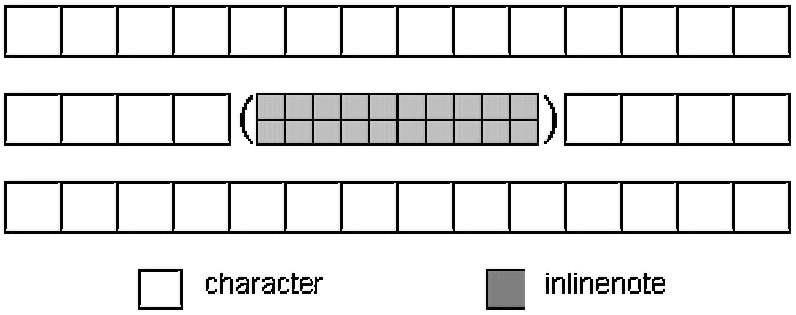

Figure 4.5 — Inlinenote

4.12 Emphasizing mark

An emphasizing mark is located at the side (right side in vertical composition or top side in horizontal composition) of a character to be emphasized. Some marking details called “mark style” should be specified for each character or a group of characters. The default mark style is “for each character”.

4.13 Superscript / Subscript (Superior / Inferior)

Superscript or subscript character is located at the side (right-top or right-bottom respectively) of an object character. In vertical composition, right-top or right-bottom is read as right-bottom or left-bottom respectively.

A size of superscript or subscript character is a half of the object character size. Both superscript and subscript characters may be located aside the same object character. Even when superscript or subscript character is located outside the virtual body of object character, the composition is based on that of inline (see 4.24).

4.14 Word-length adjustment

A word-length adjustment is employed, for example, to list several words consisting of the different number of characters. Word-lengths of all the listed words are adjusted to the same by controlling character spaces of the words as shown in Figure 4.6.

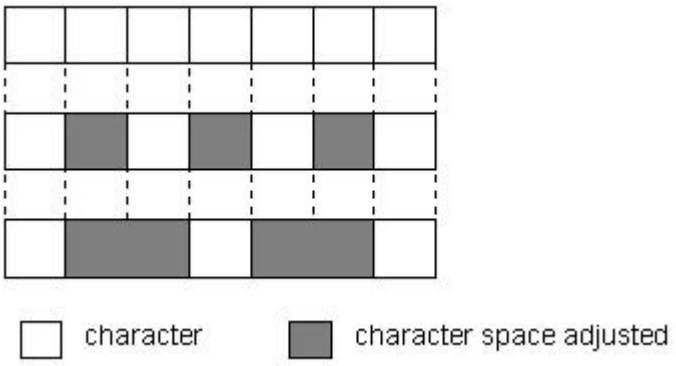

Figure 4.6 — Listed words with word-length adjustment

The total length for a word-length adjustment is specified by the product of the character size and the number of characters. A character space in each word to be adjusted is described by the expression:

character space

= [(total length specified) - (character size)(number of characters in a word)]

/ [(number of characters in a word) - 1]

4.15 Character space adjustment

In a composition of heading (see 4.19) or headline (see 3.13 and 4.8) where a line consists of comparatively small number of characters, character spaces are adjusted for readability as shown in Figure 4.7.

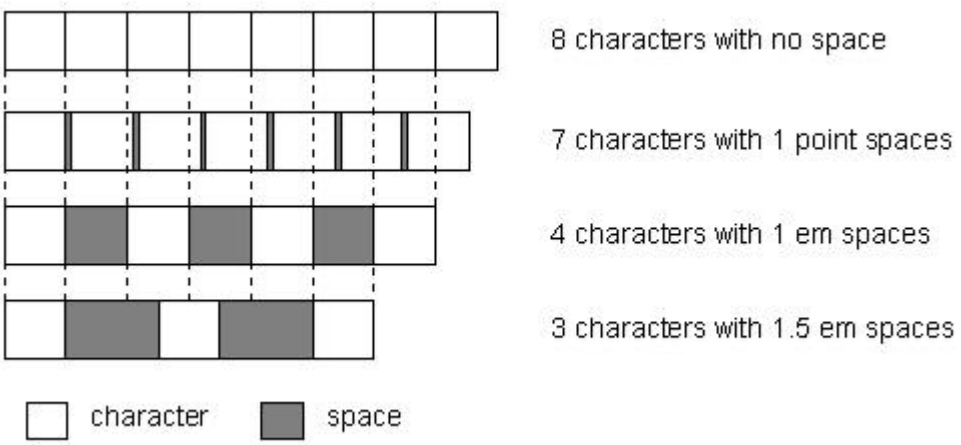

Figure 4.7 — Examples of character space adjustment

Typical character space adjustments for horizontal and vertical compositions are shown in Table 9.

Table 9 — Character spaces of typical character adjustments

Horizontal composition		Vertical composition	
Number of characters	Character space	Number of characters	Character space
2	3 em	2	2 em
3	1.5 em	3	1.25 em
4	1 em	4	3/4 em
5	1/2 em	5	1/3 em
6	1/4 em	6	1 point
7	1 point	7 or more than 7	none
8 or more than 8	none		

4.16 Clause

Clauses (including chapters and sections) are composed as:

a) ordered clause

b) unordered clause

4.16.1 Ordered clause

An ordered clause begins with:

a) Ordering character:
Latin alphabet, Kanji numerals, Arabic numerals, Roman numerals, clock numerals, etc. are used as ordered characters (see 3.20).

b) Pre/post-marking of ordering character(s):

1) pre-marking

2) pre- and post-marking

3) post-marking

c) Graphics marking

4.16.2 Unordered clause

An unordered clauses begins with:

a) No marking

b) Unordered character

c) Graphics marking
Figure 4.8 shows a beginning of an unordered clause with graphics.

Figure 4.8 — Unordered clause with graphics marking

4.16.3 Indentation of clause

For an indentation, see 4.19.

4.17 List

Lists are composed as:

a) ordered list

b) unordered list

4.17.1 Ordered list

An ordered list item begins with:

a) Ordering character:
Latin alphabet, Kanji numerals, Arabic numerals, Roman numerals, clock numerals, etc. are used as ordered characters (see 3.20).

b) Pre/post-marking of ordering character(s):

1) pre-marking

2) pre- and post-marking

3) post-marking

c) Graphics marking

4.17.2 Unordered list

An unordered list begins with:

a) No marking

b) Unordered character

c) Graphics marking

4.17.3 Indentation of list

a) indentation space on a first line of list:
One or two character indentation is employed.

b) indentation space on a second line of list:
No indentation or 1, 2, or 3 character indentation is employed.

4.18 Table

4.18.1 Character size

a) When a font family Gothic is used, texts within table and a table caption are described with the same character size.

b) When a font family Mincho is used, texts within table are described with the character size 1 point or 1 Q smaller than that of a table caption.

4.18.2 Position

a) A space between a table and image area is 1mm.

b) A space between a table and texts outside the table is 1.5 times character size of the texts.

4.19 Heading

In this composition, 2 specify models are offered following 4.19.1 and 4.19.2.

4.19.1 Character size

a) A large heading employs character sizes of 24 through 32 Q or 16 through 22 points.

b) A middle heading employs character sizes of 18 through 20 Q or 12 through 14 points.

c) A small heading employs character sizes of 14 through 16Q or 10 through 11 points.

NOTE 4 Those heading character sizes can be applied, when character sizes of texts are 8 through 9 points (12 through 13 Q).

4.19.2 Heading composition

Character sizes should be specified by using a unit of point. When using a unit of Q, see Table 1.

Some typical examples of heading compositions are shown below regarding

– Paper size A5, vertical composition, text character size 9 points and

– Paper size A5 or B5, horizontal composition, text character size 9 or 8 points.

a) Paper size A5, vertical composition, text character size 9 points

1) one heading

Type of heading	Indentation	Alignment	Illustration
large heading (14 points)	text character size 9 points×4	center of 4 lines	Figure 4.9 Left
middle heading (12 points)	text character size 9 points×6	center of 3 lines	Figure 4.9 Center
small heading (10 points)	text character size 9 points×7	center of 2 lines	Figure 4.9 Right

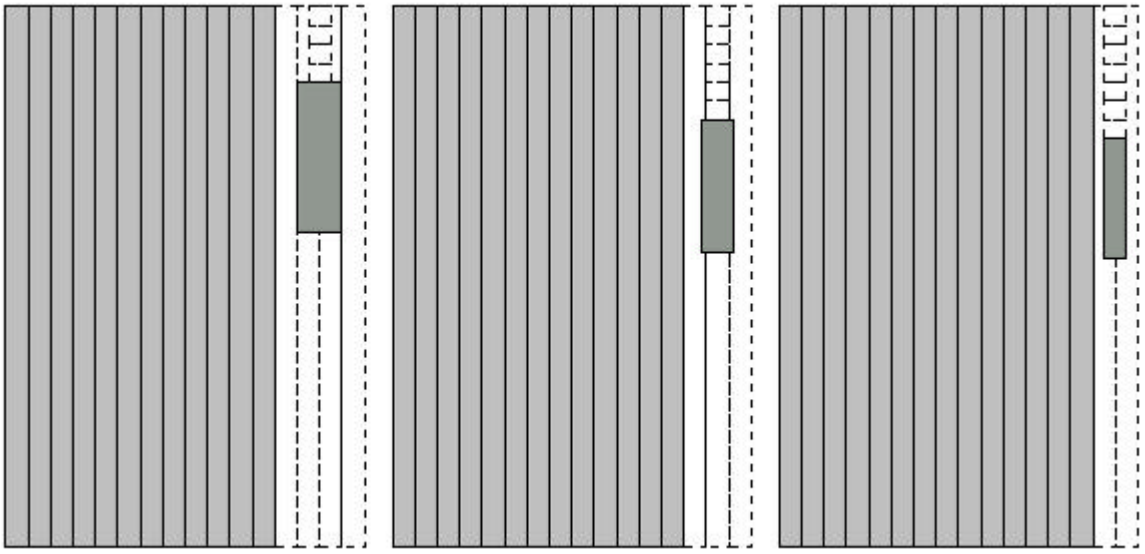

Figure 4.9 — Paper size A5, vertical composition, text character size 9 points, with one heading

2) 3 headings; large, middle, small

Type of heading	Indentation	Alignment	Illustration
large heading (14 points)	text character size 9 points×4	center of 3 lines	Figure 4.10
middle heading (12 points)	text character size 9 points×6	center of 2 lines	
small heading (10 points)	text character size 9 points×7	center of 2 lines	

8 lines space in total, including the first 1-line space.

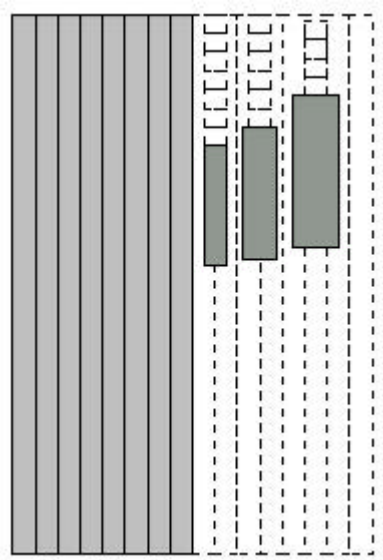

Figure 4.10 — Paper size A5, vertical composition, text character size 9 points, with 3 headings

3) 2 headings; large, middle

Type of heading	Indentation	Alignment	Illustration
large heading (14 points)	text character size 9 points×4	center of 2 lines	Figure 4.11
middle heading (12 points)	text character size 9 points×6	center of 3 lines	

6 lines space in total, including the first 1-line space.

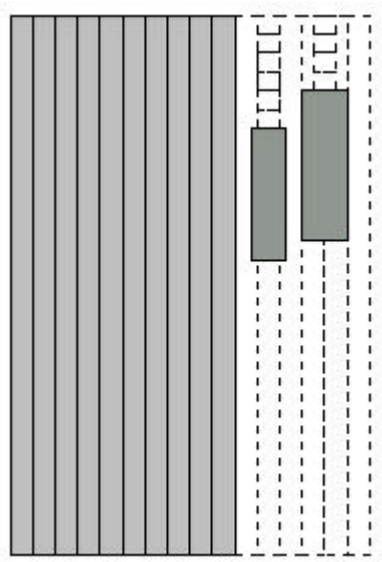

Figure 4.11 — Paper size A5, vertical composition, text character size 9 points, with 2 headings; large, middle

4) 2 headings; large, small

Type of heading	Indentation	Alignment	Illustration
large heading (14 points)	text character size 9 points×4	center of 3 lines	Figure 4.12
small heading (10 points)	text character size 9 points×7	center of 2 lines	

6 lines space in total, including the first 1-line space.

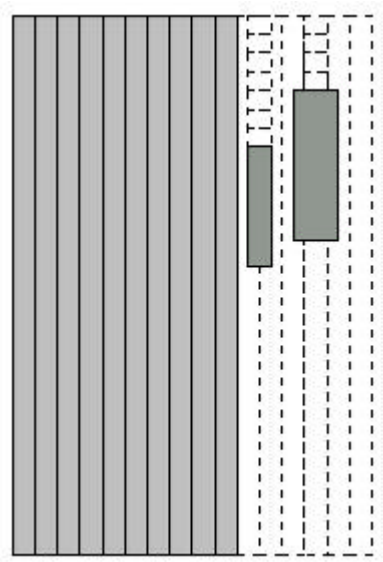

Figure 4.12 — Paper size A5, vertical composition, text character size 9 points, with 2 headings; large, small

5) 2 headings; middle, small

Type of heading	Indentation	Alignment	Illustration
middle heading (12 points)	text character size 9 points×6	center of 2 line	Figure 4.13
small heading (10 points)	text character size 9 points×7	center of 2 lines	

5 lines space in total, including the first 1-line space.

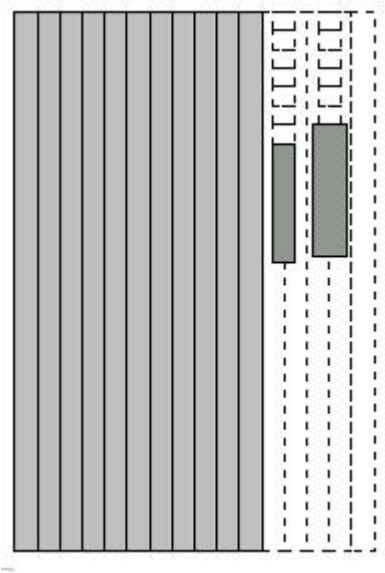

Figure 4.13 — Paper size A5, vertical composition, text character size 9 points, with 2 headings; middle, small

6) no heading with 2 lines space at the first

An illustration is shown in Figure 4.14.

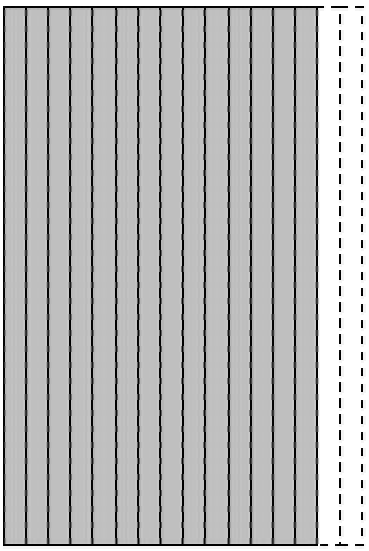

Figure 4.14 — Paper size A5, vertical composition, text character size 9 points, without heading

b) Paper size A5 or B5, horizontal composition, text character size 9 or 8 points

1) one heading

Type of heading	Indentation	Alignment	Illustration
large heading (14 points)	center of text line width	center of 4 lines	Figure 4.15 Left
middle heading (12 points)	center of text line width	center of 3 lines	Figure 4.15 Center
small heading (10 points)	text character size 9 points×1 or center of text line width	center of 2 lines	Figure 4.15 Right

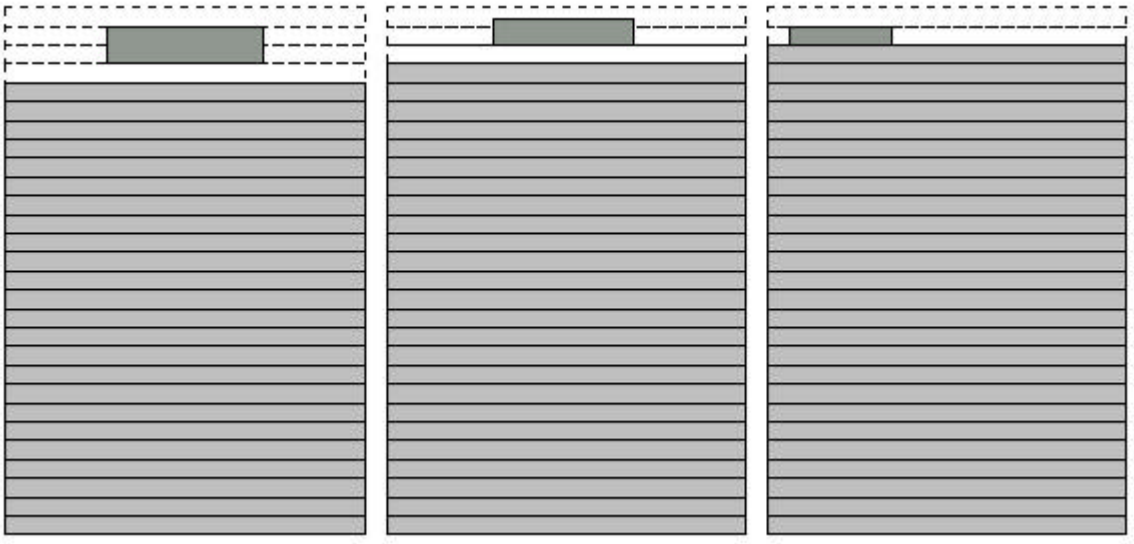

Figure 4.15 — Paper size A5 or B5, horizontal composition, text character size 9 or 8 points, with one heading

2) 3 headings; large, middle, small

Type of heading	Indentation	Alignment	Illustration
large heading (14 points)	center of text line width	center of 3 lines	Figure 4.16
middle heading (12 points)	center of text line width	center of 2 lines	
small heading (10 points)	text character size 9 points×1 or center of text line width	center of 2 lines	

8 lines space in total, including the first 1-line space.

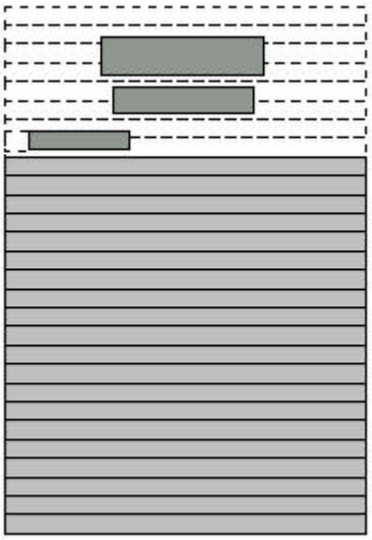

Figure 4.16 — Paper size A5 or B5, horizontal composition, text character size 9 or 8 points, with 3 headings; large, middle, small

3) 2 headings; large, middle

Type of heading	Indentation	Alignment	Illustration
large heading (14 points)	center of text line width	center of 2 lines	Figure 4.17
middle heading (12 points)	center of text line width	center of 3 lines	

6 lines space in total, including the first 1-line space.

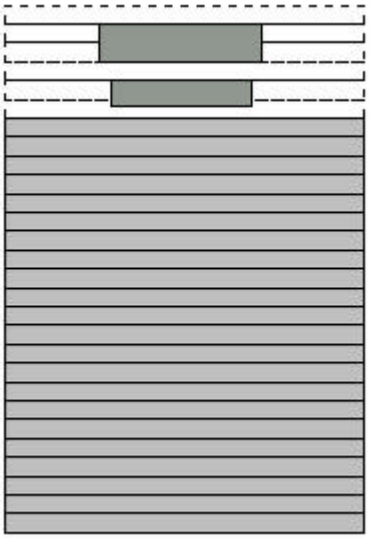

Figure 4.17 — Paper size A5 or B5, horizontal composition, text character size 9 or 8 points, with 2 headings; large, middle

4) 2 headings; large, small

Type of heading	Indentation	Alignment	Illustration
large heading (14 points)	center of text line width	center of 3 lines	Figure 4.18
small heading (10 points)	text character size 9 points×1 or center of text line width	center of 2 lines	

6 lines space in total, including the first 1-line space.

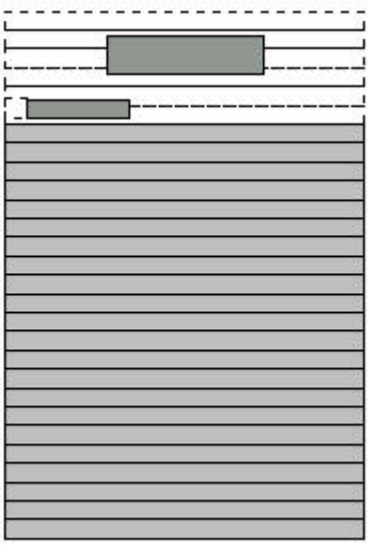

Figure 4.18 — Paper size A5 or B5, horizontal composition, text character size 9 or 8 points, with 2 headings; large, small

5) 2 headings; middle, small

Type of heading	Indentation	Alignment	Illustration
middle heading (12 points)	center of text line width	center of 2 lines	Figure 4.19
small heading (10 points)	text character size 9 points×1 or center of text line width	center of 2 lines	

5 lines space in total, including the first 1-line space.

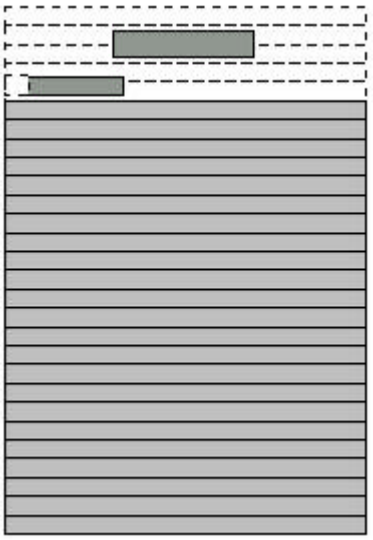

Figure 4.19 — Paper size A5 or B5, horizontal composition, text character size 9 or 8 points, with 2 headings; middle, small

4.20 Ruby

The following ruby compositions are dealt with:

a) Two-character ruby (see 3.31)

b) Ruby characters are located on the ruby line whose center corresponds to the center of object character, in vertical and horizontal composition.

c) Mono-ruby (see 3.18) and group-ruby (3.11) may be composed.

d) In mono-ruby composition, ruby characters are aligned symmetrically and with no spaces.

e) Ruby character sequence cannot be extended over the Kanji located adjacently to the object character.

f) Ruby character sequence may be extended, by one ruby character size, over the Kana located adjacently to the object character.

g) When group-ruby character sequence is shorter than the corresponding object character sequence, the spaces between ruby characters are located as shown in Figure 4.20. When group-ruby character sequence is equal to or longer than the corresponding object character sequence, no spaces between ruby characters are located.

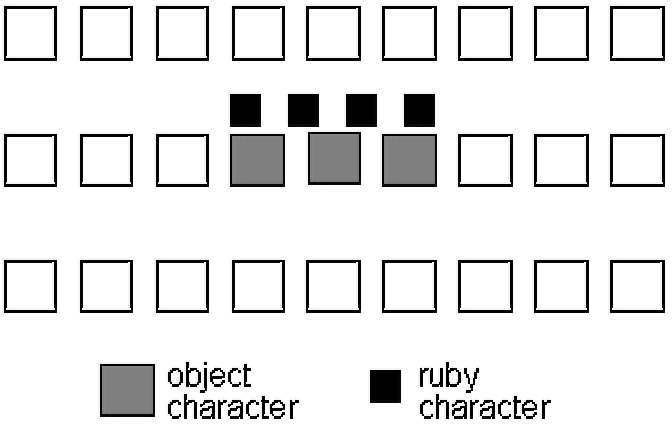

Figure 4.20 — Spacing between ruby character in grouped ruby

h) At the beginning or end of line, ruby character sequence cannot be extended beyond the object characters.

4.21 Paragraph indentation

The following paragraph indentations at the beginning of paragraphs are dealt with:

a) non-indentation

b) one-character indentation

4.22 Score

The following score compositions are dealt with:

a) underline

b) overline

c) strike-out

In vertical composition, underline/overline is read as leftline/rightline as shown in Figure 4.21.

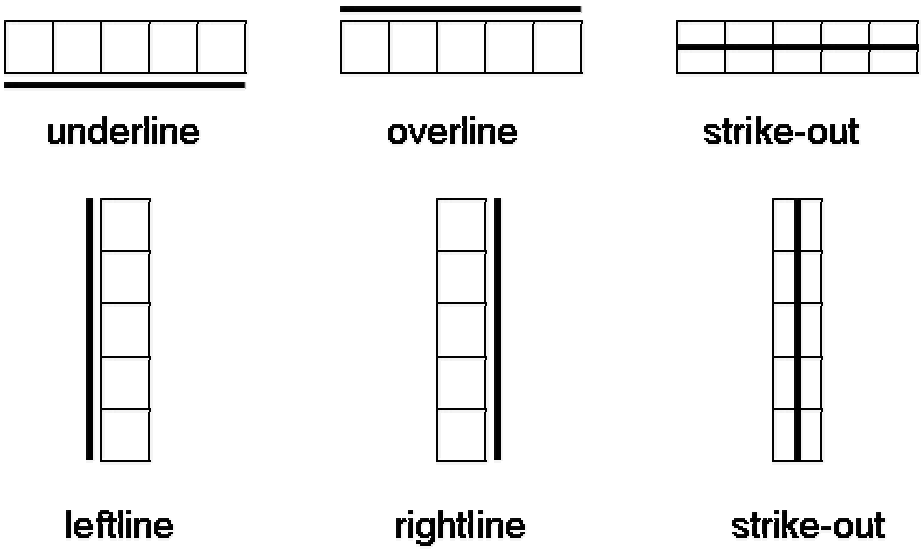

Figure 4.21 — Scores in horizontal composition (top) and vertical composition (bottom)

4.23 Rule

The rules shown in Figure 4.22 are dealt with. patterns must be only those which shown in Figure 4.22. The actual details of the rule dimensions are formatter dependent.

Rule	Rule Pattern
thin rule	
medium rule	
thick rule	
dotted rule	
dash rule	
one dotted dash rule	
two dotted dash rule	
parallel rule	
shaded rule	

Figure 4.22 — Rules and their patterns

4.24 Inline

The following elements are classified into inline elements.

a) reference mark for note

b) inlinenote

c) emphasizing mark

d) superscript and subscript

e) ruby

f) score

Those inline elements should be composed according to 4.24.1 and 4.24.2.

4.24.1 Line width

A line width should not change by including the inline element.

4.24.2 Line position

A line position should not change by including the inline element.

5 Configuration of DSSSL Library

The DSSSL library makes it feasible to describe DSSSL specifications required for complicated compositions. Configuration of the DSSSL library is shown in clause 5, and contents of the files of which the library consists are listed in clause 6 through 9.

5.1 Processing flow

The configuration and processing flow of DSSSL library is shown in Figure 5.1. In order to use the DSSSL library, Scheme processor and DSSSL processor have to be available.

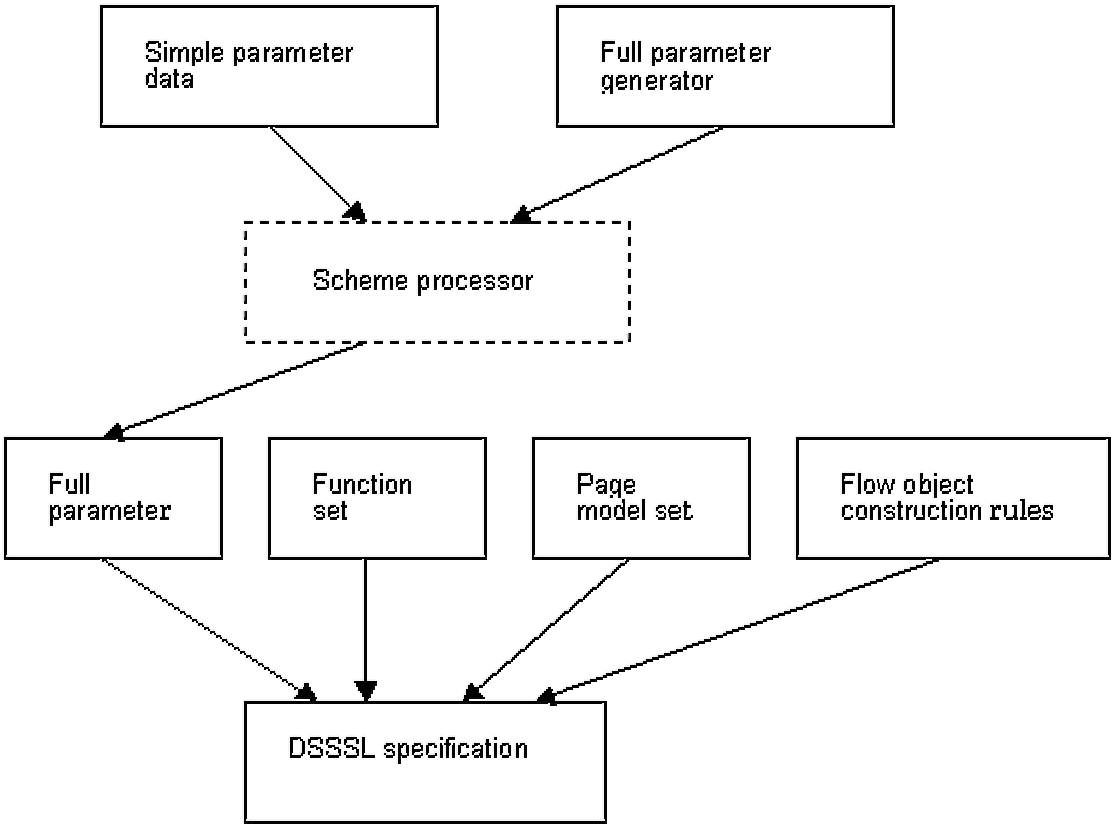

Figure 5.1 — Processing flow of the DSSSL library

The DSSSL library consists of the simple parameter data and the following 4 files:

a) full parameter generator

b) function set

c) page model set

d) flow object construction rules

5.1.1 Simple parameter data

The simple parameter data are provided in a form of association list for the full parameter generator. The parameter data are listed in 5.2.

The simple parameter data are default values for a typical composition style. If necessary, some parameter data can be added to the simple parameter data.

5.1.2 Full parameter generator

Full parameter generator creates, on a Scheme processor, full parameters being based on the simple parameter data.

5.1.3 Function set

The function set includes DSSSL flow object generating functions and their support functions used in the construction rules in DSSSL specifications. The full parameters are referred to by those functions.

5.1.4 Page model set

The page models are DSSSL descriptions of page styles. The full parameters are referred to by those page models.

5.1.5 Flow object construction rules

The construction rules provide actual DSSSL specifications for a specific DTD, using the full parameters, function set and page model set. Clause 9 shows the construction rules for HTML 3.2 DTD.

5.2 Simple parameter data

NOTE: Default unit of length is mm.

paper-name
paper size description (string, default "a4")

paper-direction
paper direction（string: "portrait"/"landscape")(default "portrait")

writing-direction
writing direction (string: "horizontal"/"vertical")(default "horizontal")

paper-width
paper width ([mm] : *paper-name* dependent)

paper-height
paper length ([mm]: *paper-name* dependent)

column-number
number of columns (number)(default 1)

base-font-size
base font size ([mm])

standard-composition
image area specification (string)

page-spec
basic page specification list (list of body font size, number of character in line, lines in page, line width, column number, space between column)

page-region-width
image area width ([mm])

page-region-height
image area height ([mm])

area-x-ratio
area vertical ratio of image area (0.0 through 1.0)(default 0.5)

area-y-ratio
area horizontal ratio of image area (0.0 through 1.0)(default 0.5)

page-region-x-offset
page region horizontal offset

page-region-y-offset
page region vertical offset

font-table
font information table of heading font (list)

ruby-font-size-factor
ruby font size ratio from base font size (0.0 through 1.0)(default 0.5)

subscript-font-size-factor
subscript font size ratio from base font size (0.0 through 1.0)(default 0.3)

superscript-font-size-factor
superscript font size ratio from base font size (0.0 through 1.0)(default 0.3)

base-font-family
base font family specification (default "mincho-light, sans-medium")

indentation-factor
number of characters of indentation of paragraph (default 1)

indent-factor
number of characters of indentation of ordered list (default 2)

has-header-page-number
existence of page number at top edge (#t/#f)(default #t)

has-header-headline
existence of headline at top edge (#t/#f)(default #t)

has-footer-page-number
existence of page number at tail edge (#t/#f)(default #f)

has-footer-headline
existence of headline at tail edge (#t/#f)(default #f)

headline-rect
headline string of rect page (string)

headline-verso
headline string of verso page (string)

footnote-number-desc
number description of foot note (string)

enum-number-desc
top character of ordered list (string)

title-number-desc
heading number description (string)

6 Full parameter generator

```
;; -*- Scheme -*-
;;
;; parameter-build.scm
;;    -- generate full parameter from spec
;;

;; -- print objects and cr.
(define (print . x)
  (map (lambda (e) (write e)(display " ")) x)
  (newline))

(define (warn . x)
  (map (lambda (e) (display e)(display " ")) x)
  (newline))

;; ------------------------------------------------
;;  storage a generated full parameters into *results*.
;; ------------------------------------------------

(define *results* '())
(define (add-result x) (set! *results* (append *results* (list x))))

;; ------------------------------------------------
;; norm-unit : normalize unit inspec
;;      unit in scheme be 'mm',
;; ------------------------------------------------

;;-- Reference 4.3
(define (Q->pt num) ;; conversion from Q to pt
  (case num
    ((7) 5)    ((8) 5.5)    ((9) 6)    ((10) 7)    ((11) 7.5)
    ((12) 8)   ((13) 9)     ((14) 10)  ((15) 10.5) ((16) 11)
    ((18) 12)  ((20) 14)    ((21) 15)  ((24) 16)  ((26) 18.5) ((28) 20)
    (else (* num (/ 0.25 0.3514)))))

(define (get-unit- nul)
  ;; devide char list into num and unit used in norm-unit
  ;;   ex) (string->list "13.5mm") --> ((#\1 #\3 #\. #\5 #\m #\m))
```

```
  (if (memq (car nul) '(#\0 #\1 #\2 #\3 #\4 #\5 #\6 #\7 #\8 #\9 #\.))
      (let ((g (get-unit- (cdr nul))))
        (cons (cons (car nul)(car g)) (cdr g)))
      (list '() nul)))

(define (rm-unit- str org)
  (let* ((x (get-unit- (string->list str)))
         (num (string->number (list->string (car x))))
         (unit (list->string (cadr x))))
    (cond ((eqv? num #f) org) ;; no number part
          ((string=? unit "mm") num) ;; It normalizes completely in mm.
          ((string=? unit "cm") (* num 10.0))
          ((string=? unit "in") (* num 25.4))
; It is 0.3514 times because it is being converted
; into mm after it is changed into pt from Q.
          ((string=? unit "Q") (* (Q->pt num) 0.3514))
          ((string=? unit "q") (* (Q->pt num) 0.3514))
          ;; 1/72 inch(American inch) is follow.
          ;;  ((string=? unit "pt") (* num (/ 25.4 72)))
          ;; a point is made "1pt = 0.3514".
          ((string=? unit "pt") (* num 0.3514))
          ((string=? unit "pica") (* num 4.233333))
          (else (warn "warning : norm-unit : unknown unit name : "
                      unit)
                num))))

;; expression with the unit is normalized in mm.
(define (norm-unit num_w_unit)
  (cond ((number? num_w_unit)
         num_w_unit)
        ((symbol? num_w_unit)
         (rm-unit- (symbol->string num_w_unit) num_w_unit))
        ((string? num_w_unit)
         (rm-unit- num_w_unit num_w_unit))))

;; function that mm makes numerical value stuck.
(define (mm x)
  (cond ((null? x) '())
        ((number? x) (string->symbol (string-append
```

```
                                (number->string x) "mm")))
        ((list? x) (cons (mm (car x))(mm (cdr x))))
        (else x)))

;; function to change into pt only targeting Q.
(define (norm-pt x)
  (let ((QtoPT
         (lambda (s x)
           (let* ((ul (get-unit- (string->list s)))
                  (num (string->number (list->string (car ul))))
                  (unit (list->string (cadr ul))))
             (cond ((eqv? num #f) x) ;; no number part
                   ((string=? unit "Q")
                    (string->symbol (string-append
                                     (number->string (Q->pt num))
                                     "pt")))
                   (else x))))))
    (cond ((null? x) '())
          ((list? x) (cons (norm-pt (car x))(norm-pt (cdr x))))
          ((symbol? x)
           (QtoPT (symbol->string x) x))
          ((string? x)
           (QtoPT x x))
          (else x))))

;  function which makes numerical value an expression with unit pt.
(define (pt x)
  (cond ((null? x) '())
  ((number? x) (string->symbol (string-append (number->string x) "pt")))
  ((list? x)(cons (pt (car x))(pt (cdr x))))
  (else x)))

;  conversion of only numerical value to pt from mm.
(define (mm->pt x)
  (round (/ x 0.3514)))

;; ------------------------------------------------
;; determine-paper-size :
;; decision of *page-width* *page-height*.
```

```
;; -----------------------------------------------
(define *paper-size-list*  ;; -- Reference 4.1
  ;;  paper name    X mm       Y mm          ;    X inch     Y inch
  '(("11x17"       (279.4     431.8)  )     ;  (11         17)
    ("a0"          (839.611   1188.16))     ;  (33.0556    46.7778)
    ("a1"          (594.078   839.611))     ;  (23.3889    33.0556)
    ("a2"          (419.806   594.078))     ;  (16.5278    23.3889)
    ("a3"          (297.039   419.806))     ;  (11.6944    16.5278)
    ("a4"          (209.903   297.039))     ;  (8.26389    11.6944)
    ("a5"          (148.519   209.903))     ;  (5.84722    8.26389)
    ("a6"          (104.775   148.519))     ;  (4.125      5.84722)
    ("a7"          (74.0833   104.775))     ;  (2.91667    4.125)
    ("a8"          (52.2111   74.0833))     ;  (2.05556    2.91667)
    ("a9"          (37.0417   52.2111))     ;  (1.45833    2.05556)
    ("a10"         (26.1056   37.0417))     ;  (1.02778    1.45833)
    ("b0"          (1000.48   1413.93))     ;  (39.3889    55.6667)
    ("b1"          (706.967   1000.48))     ;  (27.8333    39.3889)
    ("b2"          (500.239   706.967))     ;  (19.6944    27.8333)
    ("b3"          (353.483   500.239))     ;  (13.9167    19.6944)
    ("b4"          (250.119   353.483))     ;  (9.84722    13.9167)
    ("b5"          (176.742   250.119))     ;  (6.95833    9.84722)
    ("flsa"        (215.9     330.2)  )     ;  (8.5        13)
    ("flse"        (215.9     330.2)  )     ;  (8.5        13)
    ("halfletter"  (139.7     215.9)  )     ;  (5.5        8.5)
    ("ledger"      (431.8     279.4)  )     ;  (17         11)
    ("legal"       (215.9     355.6)  )     ;  (8.5        14)
    ("letter"      (215.9     279.4)  )     ;  (8.5        11)
    ("note"        (190.5     254.0)  )     ;  (7.5        10)
    ;;  ....
    ))

(define (get-paper-size paper_name)
  (assoc paper_name *paper-size-list*))

(define (determine-paper-size spec)
  (let ((pn   (assoc '*paper-name* spec))
        (dir (assoc '*paper-direction* spec))
        (flow-dir (assoc '*writing-direction* spec))
        (w    (assoc '*paper-width* spec))
```

```
    (h   (assoc '*paper-height* spec)))
(if w (set! w (norm-unit (cadr w))))
(if h (set! h (norm-unit (cadr h))))

(if (and (not (and w h)) pn)
    (let* ((pnn (cadr pn))
           (xy (assoc pnn *paper-size-list*)))
      (if xy
          (begin
           (set! xy (cadr xy))
           (if (and dir (string=? "landscape" (cadr dir)))
               (begin
                (if (not w) (set! w (cadr xy)))
                (if (not h) (set! h (car xy))))
               (begin
                (if (not w) (set! w (car xy)))
                (if (not h) (set! h (cadr xy))))))
          (warn "unknown paper name.")
          )))

(if (not (and w h))
    (determine-paper-size (cons '(*paper-name* "a4") spec))
    (begin
      (if pn
          (set! pn (cadr pn))
          (set! pn "none"))
      (add-result `(define *paper-name* ,pn))
      (if dir
          (set! dir (cadr dir))
          (set! dir "portrait"))
      (add-result `(define *paper-direction* ,dir))
      (if flow-dir
          (set! flow-dir (cadr flow-dir))
          (set! flow-dir "horizontal"))
      (add-result `(define *writing-direction* ,flow-dir))
      (add-result `(define *paper-width* ,(mm w)))
      (add-result `(define *paper-height* ,(mm h)))
      (list w h pn dir flow-dir)
      ))))
```

```
;; ------------------------------------------------
;;
;; typical composition for books.
;; (horizontal composition, landscape with a4r, etc.)
;; Default value is established.
;;
(define *standard-composition-list*
  '(;  type    paper-direct    writing-mode    colomn-n
     ("43-14"     (9pt      43   14   18pt     0))
     ("43-15"     (9pt      43   15   18pt     0))
     ("43-16"     (9pt      43   16   17pt     0))
     ("44-17"     (9pt      44   17   16pt     0))
     ("50-18"     (8pt      50   18   15pt     0))
     ("50-19"     (8pt      50   19   14pt     0)))
    (("b6" "portrait" "vertical" 2)
     ("25-20"     (8pt      25   20   14pt     2))
     ("26-20"     (8pt      26   20   14pt     2)))
    (("b6" "portrait" "horizontal" 1)
     ("30-23"     (9pt      30   23   17pt     0))
     ("33-25"     (8pt      33   25   16pt     0))
     ("33-27"     (8pt      33   27   15pt     0))
     ("34-27"     (8pt      34   27   15pt     0)))

    (("b5" "portrait" "vertical" 1)
     ("24-31"     (8pt      24   31   13pt     0)))
    (("b5" "portrait" "horizontal" 1)
     ("43-32"     (9pt      43   32   18pt     0))
     ("regular"   (13Q      42   31   26Q      0)))
    (("b5" "portrait" "horizontal" 2)
     ("23-44"     (9pt      23   44   14pt     2))
     ("22-41"     (9pt      22   41   15pt     2))
     ("25-51"     (8pt      25   51   12pt     2))
     ("regular "  (13Q      22   43   20Q      2))
     ("wide"      (13Q      21   39   22Q      2))
     ("small"     (12Q      23   48   18Q      2)))

    (("a6" "portrait" "vertical" 1)
     ("41-13"     (8pt      41   13   17pt     0))
```

```
 ("41-14"     (8pt      41   14   16pt     0))
 ("41-15"     (8pt      42   15   15pt     0))
 ("42-13"     (8pt      42   13   16pt     0))
 ("42-14"     (8pt      42   14   16pt     0))
 ("42-15"     (8pt      42   15   15pt     0))
 ("43-15"     (8pt      43   15   15pt     0))
 ("43-16"     (8pt      43   16   14pt     0))
 ("43-18"     (8pt      43   18   13pt     0))
 ("43-19"     (8pt      43   19   13pt     0)))

(("a5" "portrait" "vertical" 1)
 ("51-16"     (9pt      51   16   18pt     0))
 ("52-16"     (9pt      52   16   18pt     0))
 ("52-17"     (9pt      52   17   18pt     0))
 ("52-18"     (9pt      52   18   17pt     0))
 ("52-19"     (9pt      52   19   17pt     0)))
(("a5" "portrait" "vertical" 2)
 ("25-20"     (9pt      25   20   15pt     2))
 ("30-24"     (8pt      30   24   13pt     2))
 ("29-23"     (8pt      29   23   14pt     2)))
(("a5" "portrait" "horizontal" 1)
 ("35-26"     (9pt      35   26   18pt     0))
 ("35-28"     (9pt      35   28   17pt     0))
 ("35-30"     (9pt      35   30   16pt     0))
 ("40-30"     (8pt      40   30   16pt     0))
 ("38-33"     (8pt      38   33   14pt     0))
 ("regular"   (13Q      34   27   25Q      0))
 ("narrow"    (13Q      34   29   23Q      0))
 ("small"     (12Q      37   28   24Q      0))
 ("narrow-small"  (12Q  36   31   21Q      0)))

;; the following a4 is not a standard value.
(("a4" "portrait" "horizontal" 1)
 ("regular"  (13Q       51    41   24Q      0))
 ("narrow"    (14Q       48    39   25Q      0)))
(("a4" "portrait" "horizontal" 2)
 ("regular"  (14Q       24    42   23Q      2)))
(("a4" "portrait" "horizontal" 3)
 ("regular"  (14Q       16    42   23Q      2)))
```

```
))

;; a party direction by horizontal writing.
(define (calc-xx cn xx fd)
  (let ((fs  (norm-unit (car xx)))
        (num (norm-unit (cadr xx)))
        (ln  (norm-unit (caddr xx)))
        (lw  (norm-unit (cadddr xx)))
        (cs  (cddddr xx)))
    (if (= (length cs) 0)
        (set! cs 0) ;; if cs is omitted
        (set! cs (car cs)))
    (let ((rw (+ (* cn num fs) (* (- cn 1) cs fs)))
          (rh (* ln lw)))
      (if (string=? fd "horizontal")
          (list rw rh)
          (list rh rw)) )))

(define (get-sc pn dir flow-dir cn sc)
  (let ((e (list pn dir flow-dir cn))(ret #f))
    (set! ret (assoc e *standard-composition-list*))
    (if ret
        (begin
          (set! ret(cdr ret))
          (set! ret (assoc sc ret))
          (if ret
              (cadr ret) ;; return xx
              #f))
        #f)))

;;  *page-spec* is generated when value is incorrect
;;  *standard-composition* isn't given.
(define (get-xx pn dir flow-dir cn )
  (let ((e (list pn dir flow-dir cn))
        (ret #f)
        (sc "default"))
    (set! ret (assoc e *standard-composition-list*))
    (if ret
    (begin
```

```
      (set! ret (cdr ret))
      (set! ret (car ret))
      (cadr ret))
 #f)))

(define (determine-regionsize ps spec)
  (let (
        (w   (car ps))
        (h   (cadr ps))
        (pn  (caddr ps))
        (dir (cadddr ps))
        (flow-dir (cadddr (cdr ps)))
        (cn  (assoc '*column-number* spec))
        (xx  (assoc '*page-spec* spec)) ;; (FontSZ Letters Lines Feeds Colomn space)
        (rw  (assoc '*page-region-width* spec))
        (rh  (assoc '*page-region-height* spec))
        (sc  (assoc '*standard-composition* spec))
        (fs  (assoc '*base-font-size* spec))
        (xr  (assoc '*area-x-ratio* spec))
        (yr  (assoc '*area-y-ratio* spec))
        (xroff (assoc '*page-region-x-offset* spec))
        (yroff (assoc '*page-region-y-offset* spec))
        (x-off 0)
        (y-off 0)
        )

    (if fs (set! fs (cadr fs)))
    (if cn (set! cn (cadr cn)))
    (if (not (number? cn)) (set! cn 1))
    (if rw (set! rw (norm-unit (cadr rw))))
    (if rh (set! rh (norm-unit (cadr rh))))
    (if sc (set! sc (cadr sc)))
    (if xx (set! xx (cadr xx)))

    ;;;; typical compositions.
    ;;
    ;; (1)  Column set  + *page-region-width*/*page-region-height*
    ;;
    ;; (2)  Column set  + (x x x x x)
```

```
;;
;; (3)  Paper size/direction
;;      writing-mode + *standard-composition*
;;      Colomn set
;;
;; one of them is required. ((1) high priority)
;;
;; default number of column, single-column-set (*column-number* == 1)
;;

(if (and (not xx) sc) (set! xx (get-sc pn dir flow-dir cn sc)))
(if xx
    (begin
      (set! xx (norm-pt xx))
      (let ;; pattern (2) or (3)
          ((ret (calc-xx cn xx flow-dir)))
        (if ret
            (begin
              (if (not rw) (set! rw (car ret)))
              (if (not rh) (set! rh (cadr ret)))
              )))))

(if cn (add-result `(define *column-number* ,cn)))
(if xx (add-result `(define *page-spec* ',xx)))
(if sc (add-result `(define *standard-composition* ,sc)))

;; function defined when no region height/width is determined.
    (if (not rw)
    (begin
      (warn "no region width.")
      (set! rw (* w 0.8))))
(if (not rh)
    (begin
      (warn "no region height.")
      (set! rh (* h 0.8))))

(add-result `(define *page-region-width* ,(mm rw)))
(add-result `(define *page-region-height* ,(mm rh)))
```

```
  (if (not xx)
  (begin
    (set! xx (get-xx pn dir flow-dir cn))
    (if sc (warn "*standard-composition* error"))
    (if xx
        (begin
      (add-result `(define *page-spec* ,xx))
      (warn "default *page-spec*"))
                  (add-result `(undefine *page-spec* #f)))))

  ;;;;;;;;; Position of image area
  ;;
  ;;  when *page-region-x-offset* /
  ;;  *page-region-y-offset* are given, they specify the position of image area.
  ;;  Otherwise, it is the ratio
  ;;    *area-x-ratio* / *area-y-ratio*.
  ;;    default ratio is 0.5.
  ;;
  (if (not xr) (set! xr 0.5))
  (if (not yr) (set! yr 0.5))
  (add-result `(define *area-x-ratio* ,xr))
  (add-result `(define *area-y-ratio* ,yr))
  (if    xroff
      (set! x-off xroff)
      (set! x-off (* (- w rw) xr)))
  (if    yroff
      (set! y-off yroff)
      (set! y-off (* (- h rh) yr)))

  (add-result `(define *page-region-x-offset* ,(mm x-off)))
  (add-result `(define *page-region-y-offset* ,(mm y-off)))

  (if (not fs) (if xx (set! fs (car xx))))
  (if fs (determine-font-size fs spec))

  (list x-off y-off)
  ))

(define (assoc-value e l)
```

```
  (let ((ret (assoc e l)))
    (if ret (cdr ret) ret)))

;;
;; Headline character size
;;
;; -- reference 4.19
(define *default-font-table*
  '( (("a5" "portrait" "vertical" (9pt))
      (s   0 (large  14pt 4 4) (medium 12pt 6 3) (small  10pt 7 2))
      (lms 1 (large  14pt 4 3) (medium 12pt 6 2) (small  10pt 7 2))
      (lm  1 (large  14pt 4 2) (medium 12pt 6 3))
      (ls  1 (large  14pt 4 3)                   (small  10pt 7 2))
      (ms  1                   (medium 12pt 6 2) (small  10pt 7 2))
      (n   2))
     (("a5" "portrait" "horizontal" (9pt 8pt))
      (s   0 (large  14pt 'c 4) (medium 12pt 'c 3) (small  10pt 'c 2))
      (lms 1 (large  14pt 'c 3) (medium 12pt 'c 2) (small  10pt 'c 2))
      (lm  1 (large  14pt 'c 2) (medium 12pt 'c 3))
      (ls  1 (large  14pt 'c 3)                    (small  10pt 'c 2))
      (ms  1                   (medium 12pt 'c 2) (small  10pt 'c 2))
      (n   2))))

; default *font-table*
(define (determine-font-size fs flow-dir spec)
  (if fs (add-result `(define *base-font-size* ,fs)))
  (let ((jtm (assoc '*font-table* spec)))
    (if jtm
        (add-result `(define *font-table* ',jtm))
       ; for later feasibility, "vertical", "horizontal" are judged,
       ; and default *font-table* is returned.
        (begin
          (if (string=? flow-dir "vertical")
              (set! jtm (cdr (list-ref *default-font-table* 0)))
              (set! jtm (cdr (list-ref *default-font-table* 1))))
          (add-result `(define *font-table*',jtm)))
        ))) ;; --- need to fix .....

;; default items/chapter number/footnote number
```

```
(define *default-footnote-number-desc*
  '(#f #f "" "" ")")
)

(define *default-enum-number-desc*
  '(((1) #f "(" "" ")")   ;; (1)
    ((2) 'ABC "" "" ".")  ;;   A.
    ((3) 'abc "" "" ")")  ;;    a)
))

(define *default-title-number-desc*
  '(((1) #f "第" "." "章")  ;; 第 1 章
    ((2) #f     ""   "." "節")  ;; 1.1
    ((3) #f     ""   "." "項")  ;; 1.1.1
    ((n) #f     ""   "." ""))   ;; 1.1.1.1
)

(define (check-other-defs spec)
  (let ((rubi-f (assoc-value '*rubi-font-size-factor* spec))
        (subs-f (assoc-value '*subscript-font-size-factor* spec))
        (sups-f (assoc-value '*superscript-font-size-factor* spec))
    (bfw    (assoc-value '*base-font-weight* spec))
    (bfp    (assoc-value '*base-font-posture* spec))
        (bff    (assoc-value '*base-font-family* spec))
;       (tff    (assoc-value '*title-font-family* spec))
        (findf  (assoc-value '*jisage-factor* spec))
        (indf   (assoc-value '*indent-factor* spec))

        (hhn    (assoc-value '*has-header-nonburu* spec))
        (hhh    (assoc-value '*has-header-hasira*  spec))
        (hfn    (assoc-value '*has-footer-nonburu* spec))
        (hfh    (assoc-value '*has-footer-hasira*  spec))
        (hr     (assoc-value '*hasira-rect* spec))
        (hv     (assoc-value '*hasira-verso* spec))
        (fs     (assoc-value '*footnote-style* spec))
        (fe     (assoc-value '*footnote-exp* spec))
        (fn-nd  (assoc-value '*footnote-number-desc* spec))
        (en-nd  (assoc-value '*enum-number-desc* spec))
        (tt-nd  (assoc-value '*title-number-desc* spec))
```

```
        )
    (add-result `(define *rubi-font-size-factor* ,(if rubi-f (car rubi-f) 0.5)))
    (add-result `(define *subscript-font-size-factor*
                  ,(if subs-f (car subs-f) 0.3)))
    (add-result `(define *superscript-font-size-factor*
                  ,(if sups-f (car sups-f) 0.3)))
    (add-result `(define *base-font-weight*
                  ',(if bfw (car bfw) 'medium)))
    (add-result `(define *base-font-posture*
                  ',(if bfp (car bfp) 'upright)))
    (add-result `(define *base-font-family*
                   ',(if bff (car bff) "mincho-light,sans-medium"))) ;; The default of the font name.
;    (add-result `(define *title-font-family*
;                  ',(if tff (car tff) "gochic-light,times-medium"))) ;; The default of the font name.
    (add-result `(define *jisage-factor*
                  ,(if findf (car findf) 1))) ;; The number of the characters
;; that a letter is indented in the beginning of the paragraph.
    (add-result `(define *indent-factor*
                  ,(if indf (car indf) 2))) ;; The number of the characters that a letter is indented.
    (add-result `(define *has-header-nonburu* ,(if hhn (car hhn) #t)))
    (add-result `(define *has-header-hasira*  ,(if hhh (car hhh) #t)))
    (add-result `(define *has-footer-nonburu* ,(if hfn (car hfn) #f)))
    (add-result `(define *has-footer-hasira*  ,(if hfh (car hfh) #f)))
    (add-result `(define *hasira-rect*  ,(if hr (car hr) " ")))
    (add-result `(define *hasira-verso*  ,(if hv (car hv) " ")))
    (add-result `(define *footnote-exp*  ,(if fe (car fe) "[]")))
    (add-result `(define *footnote-number-desc*
                  ,(if fn-nd (car fn-nd)
                       *default-footnote-number-desc*)))
    (add-result `(define *enum-number-desc*
                  ,(if en-nd (car en-nd)
                       *default-enum-number-desc*)))
    (add-result `(define *title-number-desc*
                  ,(if tt-nd (car tt-nd)
                       *default-title-number-desc*)))
))


;;
```

```
;; top function
;;
(define (build-page page-spec)

;  (set! *results* '())
  (set! *results* '( ;; It is defined in advance, of the spider.
                    (define-unit Q 0.25mm)
                    (define-unit pt 0.3514mm)
                    ;(define-unit pi (/ 1in 6))
                    ;(define-unit pt (/ 1in 72))
                    ;(define-unit px (/ 1in 96))
                  ))
  (let* ((wh  (determine-paper-size page-spec)))
    (determine-region size wh page-spec)
    (check-other-defs page-spec)
    *results*
))

;(define (r) (pp *results*))
(define (r)
  (let loop ((rs *results*))
    (cond ((null? rs) #t)
          (else
           (print (car rs))
           (loop (cdr rs))))))

;;;;; sample
;
(define page-spec
  '(
    (*paper-name* "b5")
    (*paper-direction* "portrait") ;; "landscape" / "portrait"
    (*standard-composition* "standard")
    (*column-number* 1)
    ))
;
(build-page page-spec)
(r)
;
```

7 Function set

```
;; -*- Scheme -*-
;;              Amended Functions.dsl

;
; Numbering Functions for section and footnote
;

(define (char-repeat sym num)
  (string-append (char-repeat sym (- num 1)) sym))

(define (num->alpha num)
  (substring " abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"
             num (+ num 1)))

(define (num->alphaCAP num)
  (substring " ABCDEFGHIJKLMNOPQRSTUVWXYZ"
             num (+ num 1)))

(define (num->kanji num) ;; Assume max 2 digit.
  (if (> num 10)
      (string-append (if (= (quotient num 10) 1)
                         ""
                         (num->kanji (quotient num 10)))
                     "十"
                     (num->kanji (remainder num 10)))
      (case num
        ((0) "") ((1) "一") ((2) "二") ((3) "三") ((4) "四") ((5) "五")
        ((6) "六") ((7) "七") ((8) "八") ((9) "九") ((10) "十"))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Roma numbering function
(define (num->roma num)
  (format-number num "i"))

(define (make-num n exp)
  (case exp
    (('abc) (num->alpha n))
    (('ABC) (num->alphaCAP n))
```

```
    (('kanji) (num->kanji n))
    (('roma) (num->roma n))
;   (('hira) (num->hira n))
;   (('kata) (num->hira n))
;   (('iroha-hira) (num->hira n))
;   (('iroha-kata) (num->hira n))
    (('asterisk)  (char-repeat "*" n))
    (('dag) (char-repeat " † " n))  ; Dag, ddag, P, S and Vert follow TeX.
    (('ddag) (char-repeat " ‡ " n))
    (('P) (char-repeat " ¶ " n))
    (('S) (char-repeat " § " n))
    (('Vert) (char-repeat " ‖ " n))
    (('sharp)  (char-repeat "#" n))
    (else (number->string n))
))

(define (make-numbering nl desc)
(if (equal? #f desc)
  (make-numbering nl *title-number-desc*)
  (letrec ((num-member
            (lambda (e l)
              (cond ((not(list? l)) #f)
                    ((member e l) #t)
                    ((eq? 'nl (car (reverse l)))
                     (let* ((ll (reverse (cdr (reverse l)))))
                       (if (> (length ll) 0)
                           (> e (apply max ll))
                           #t)))
                    (else #f))))
           (match-level
            (lambda (level desc)
              (cond ((null? desc) #f)
                    ((not (list? (car desc))) desc)
                    ((not (list? (caar desc))) (car desc)) ; non-list
                    ((num-member level (caar desc)) (car desc))
                    (else (match-level level (cdr desc)))
                    )))
           (make-infix
            (lambda (nl num infix)
```

```
;            (cond ((eqv? 'last infix)                to be revised
             (cond ((or (eqv? 'last infix) (eqv? "" infix))
                    (make-num (car (reverse nl)) num))
                   ((null? (cdr nl))
                    (make-num (car nl) num))
                   (else
                    (string-append
                     (make-num (car nl) num)
                     infix
                     (make-infix (cdr nl) num infix)))))))
    (cond ((number? nl) (make-numbering (list nl) desc))
          ((not (list? nl)) "")
          ((not (list? desc)) "")
          ((match-level (length nl) desc)
;           => (lambda (d)                          to be revised
            (let ((d (match-level (length nl) desc)))
              (string-append
               (caddr d) ;; pre
               (make-infix nl (cadr d)(cadddr d))
               (cadddr (cdr d)) ;; post
               )))
          (else "")))))

;; Paragraph

(define *paragraph-style* ;; Base set of paragraph
  (style
   font-size: *base-font-size*
   font-weight: *base-font-weight*
   font-posture: *base-font-posture*
   font-family-name: *base-font-family*
   line-spacing: (cadddr *page-spec*)
   quadding: 'start ;; Left alignment.
   ))

(define *fli-paragraph-style* ;; Indented paragraph
  (style
   use: *paragraph-style*
   first-line-start-indent: (* *base-font-size* *jisage-factor*)
```

```
  ))

(define *indent-step*
  (* *base-font-size* *indent-factor*))


;; Headline/page number
;;     - See PAGE-HEADER and PAGE-FOOTER,  Here only style.

(define *header-footer-style*
  (style
   use: *paragraph-style*
   font-size: (* *base-font-size* 1.0)
   line-spacing: (* (cadddr *page-spec*) 1.0)
   ;;font-posture: 'italic

   ))

;; Footnote

(define *footnote-style*
  (style
   use: *paragraph-style*
   font-size: (* *base-font-size* 1.0)
   line-spacing: (* (cadddr *page-spec*) 1.0)
   ))

;; word-length adjustment

(define (jidori n)
  (make line-field
        field-width: (* *base-font-size* n) ;; n jidori
        (make paragraph
              quadding: 'justify
              last-line-quadding: 'justify
              (process-children))))

;; Caption
```

```
;;; node count function

(define (GET-MIDASHI-NUMS node-list)
;   (make-numbering (element-number-list node-list)))     to be revised
    (make-numbering (element-number-list node-list) #f))

;; Ruby

;;;;;;; improvement (glyph-annotation) ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;(define (RUBI)
;  (make glyph-annotation                                    to be revised
;        annotation-glyph-placement: 'centered
;        annotation: (process-matching-children 'yomi)
;        (process-children)))
;
;(define (YOMI . f)
;  (let ((factor (if (null? f) *ruby-font-size-factor* (car f))))
;    (make paragraph
;          font-size: (* (inherited-font-size) factor)
;          (process-children))))

;;;;;; (glyph-annotation) ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(define (YOMI)
  (sosofo-append
    (literal "(")
    (process-children)
    (literal ")")))

;; superscript / subscript

;     period allows to omit the argument.
;     processor does not support the syntax.

;(define (SUBSCRIPT . f)                          to be revised
(define (SUBSCRIPT   f)
  (let ((factor (if (null? f) *subscript-font-size-factor* (car f))))
    (make math-sequence
          math-display-mode: 'inline
          (make subscript
```

```
            font-size: (* (inherited-font-size) factor)
            (process-children)))))

;(define (SUPERSCRIPT . f)                          to be revised
(define (SUPERSCRIPT   f)
  (let ((factor (if (null? f) *superscript-font-size-factor* (car f))))
    (make math-sequence
          math-display-mode: 'inline
          (make superscript
                font-size: (* (inherited-font-size) factor)
                (process-children)))))

;; Underline

(define (UNDERLINE)
  (make score
        type: 'after
        (process-children)))

;; Inlinenote

;;;;;; improvement (multi-line-inline-node) ;;;;;;;;;;;;;;;;;;;;;;;;;;;
;(define (WARICHUU)
;  (make multi-line-inline-node ;; see ISO/IEC10179 12.6.24          to be revised
;        (process-children)))

;; Emphasized mark

;;;;;; improvement (emphasizing-mark) ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;(define (KENTEN)
;  (make emphasizing-mark ;; see ISO/IEC10179 12.6.25              to be revised
;        mark: (literal " • ")
;        (process-children)))

;; Font / Typeface

(define (BOLD-SEQ)
  (make sequence
    font-weight: 'bold
```

```
    (process-children)))

(define (ITALIC-SEQ)
  (make sequence
    font-posture: 'italic
    (process-children)))

(define (BOLD-ITALIC-SEQ)
  (make sequence
        font-weight: 'bold
        font-posture: 'italic
        (process-children)))

(define (STRIKE-SEQ)
  (make score
    type: 'through
    (process-children)))

;; List

(define (MAKE-ENUM-EXP nul)
  (make-numbering nul *enum-number-desc*))


;;;;;;; to be revised ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;(define (LIST-CONTAINER)
;  (make display-group
;        ;space-before:
;        ;space-after:
;        start-indent: (inherit-start-indent)
;))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(define (LIST-CONTAINER)
  (make display-group
        ;space-before:
        ;space-after:
        start-indent: (+ (inherited-start-indent) (* *indent-factor* *base-font-size*))
))
```

```
;;;;;;; to be revised ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;(define (LIST-ELEMENT lhead)  ;; lhead -- list head string
;  (make paragraph
;        use: *paragraph-style*
;        ;space-before:
;        start-indent: (+ (inherit-start-indent)
;                         (* *indent-factor* *base-font-size*))
;        first-line-start-indent: (- (* *indent-factor* *base-font-size*))
;        (make line-field
;              field-width: (* *indent-factor* *base-font-size*)
;              (literal lhead))
;        (process-children-trim)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(define (LIST-ELEMENT lhead)  ;; lhead -- list head string
  (make paragraph
        use: *paragraph-style*
        ;space-before:
        start-indent: (inherited-start-indent)
        first-line-start-indent: (- (* *indent-factor* *base-font-size*))
        (make line-field
              field-width: (* *indent-factor* *base-font-size*)
              (literal lhead))
        (process-children-trim)))

;; Title

;;;;;;; to be revised ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;(define (TITLE-LARGE)
;  (make paragraph
;        use: *paragraph-style*
;        font-size: (list-ref *font-table* 2)
;        font-weight: 'bold
;        space-before:
;        (car
;         (case (gi (first (children (current-node))))
;           (("h2" "h3") (car (title-vertical-spacing "h1" 2)))
;           (else  (car (title-vertical-spacing "h1" 0)))))
```

```
;       space-after:
;       (cadr
;        (case (gi (first (children (current-node))))
;          (("h2" "h3") (cadr (title-vertical-spacing "h1" 2)))
;          (else  (cadr (title-vertical-spacing "h1" 0)))))
;       escapement-space-after: ;; used by character flow object class.
;       (title-char-spacing (count (children (current-node))) *paper-name*)
;       (literal (GET-MIDASHI-NUMS '("H1")))
;       (literal "    ")
;       (process-children)))

;;;;;;; to be revised ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;(define (TITLE-MEDIUM)
;  (make paragraph
;       use: *paragraph-style*
;       font-size: (list-ref *font-table* 3)
;       font-weight: 'bold
;       space-before:
;       (let ((b1 (absolute-first-sibling? (current-node)))
;             (b2 (string=? (gi (first (children (current-node)))) "h3")))
;         (car
;          (cond
;           ((and b1 b2) (title-vertical-spacing "h2" 3))
;           ((and (not b1) b2) (title-vertical-spacing "h2" 2))
;           ((and b1 (not b2)) (title-vertical-spacing "h2" 1))
;           (else (title-vertical-spacing "h2" 0)))))
;       space-after:
;       (let ((b1 (absolute-first-sibling? (current-node)))
;             (b2 (string=? (gi (first (children (current-node)))) "h3")))
;         (cadr
;          (cond
;           ((and b1 b2) (title-vertical-spacing "h2" 3))
;           ((and (not b1) b2) (title-vertical-spacing "h2" 2))
;           ((and b1 (not b2)) (title-vertical-spacing "h2" 1))
;           (else (title-vertical-spacing "h2" 0)))))
;       escapement-space-after:
;       (title-char-spacing (count (children (current-node))) *paper-name*)
; ;     (make embedded-text   ;; Title number                  to be revised
;             direction: 'right-to-left
```

```
;               escapement-space-after: 0
;               (literal (GET-MIDASHI-NUMS '(H1 H2)))
;               (literal "    ")
;                )
;       (process-children)))

;;;;;;; to be revised ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;(define (TITLE-SMALL)
;  (make paragraph
;       use: *paragraph-style*
;       font-size: (list-ref *font-table* 4)
;       font-weight: 'bold
;       space-before:
;       (car
;        (if (absolute-first-sibling? (current-node))
;            (title-vertical-spacing "h2" 1)
;            (title-vertical-spacing "h2" 0)))
;       space-after:
;       (cadr
;        (if (absolute-first-sibling? (current-node))
;            (title-vertical-spacing "h2" 1)
;            (title-vertical-spacing "h2" 0)))
;       escapement-space-after:
;       (title-char-spacing (count (children (current-node))) *paper-name*)
; ;      (make embedded-text   ;; Title number                to be revised
;             direction: 'right-to-left
;             escapement-space-after: 0
;             (literal (GET-MIDASHI-NUMS '(H1 H2 H3)))
;             (literal "    ")
;              )
;       (process-children)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(define (TITLE-LARGE)
  (make paragraph
       use: *paragraph-style*
       font-size: (list-ref (list-ref *font-table* 1) 1)
       font-weight: 'bold
       space-before:
```

```
        (car
         (case (gi (first (children (current-node))))
           (("h2" "h3") (title-vertical-spacing "h1" 2))
           (else (title-vertical-spacing "h1" 0))))
        space-after:
        (cadr
         (case (gi (first (children (current-node))))
           (("h2" "h3") (title-vertical-spacing "h1" 2))
           (else (title-vertical-spacing "h1" 0))))
        escapement-space-after:
           (title-char-spacing (count (children (current-node))) *paper-name*)
        (literal (GET-MIDASHI-NUMS '("H1")))
    (literal "  ")
        (process-children)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(define (TITLE-MEDIUM)
  (make paragraph
        use: *paragraph-style*
        font-size: (list-ref (list-ref *font-table* 2) 1)
        font-weight: 'bold
        space-before:
        (let ((b1 (absolute-first-sibling? (current-node)))
              (b2 (string=? (gi (first (children (current-node)))) "h3")))
          (car
           (cond
            ((and b1 b2) (title-vertical-spacing "h2" 3))
            ((and (not b1) b2) (title-vertical-spacing "h2" 2))
            ((and b1 (not b2)) (title-vertical-spacing "h2" 1))
            (else (title-vertical-spacing "h2" 0)))))
        space-after:
        (let ((b1 (absolute-first-sibling? (current-node)))
              (b2 (string=? (gi (first (children (current-node)))) "h3")))
          (cadr
           (cond
            ((and b1 b2) (title-vertical-spacing "h2" 3))
            ((and (not b1) b2) (title-vertical-spacing "h2" 2))
            ((and b1 (not b2)) (title-vertical-spacing "h2" 1))
            (else (title-vertical-spacing "h2" 0)))))
```

```
        escapement-space-after:
        (title-char-spacing (count (children (current-node))) *paper-name*)
        (literal (GET-MIDASHI-NUMS '("H1" "H2")))
        (literal "  ")
        (process-children)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(define (TITLE-SMALL)
  (make paragraph
        use: *paragraph-style*
        font-size: (list-ref (list-ref *font-table* 3) 1)
        font-weight: 'bold
        space-before:
        (car
         (if (absolute-first-sibling? (current-node))
             (title-vertical-spacing "h2" 1)
             (title-vertical-spacing "h2" 0)))
        space-after:
        (cadr
         (if (absolute-first-sibling? (current-node))
             (title-vertical-spacing "h2" 1)
             (title-vertical-spacing "h2" 0)))
        escapement-space-after:
        (title-char-spacing (count (children (current-node))) *paper-name*)
        (literal (GET-MIDASHI-NUMS '("H1" "H2" "H3")))
        (literal "  ")
        (process-children)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Some additional definitions
(define (caar a)
    (car (car a)))

(define (cadr a)
    (car (cdr a)))

(define (caddr a)
    (car (cdr (cdr a))))
```

```
(define (cadddr a)
    (car (cdr (cdr (cdr a)))))

(define (first a)
    (node-list-first a))

(define (eq? a b)
        (if (and (not (list? a)) (not (list? b)))
        (equal? a b)
        #f))

(define (eqv? a b)
    (eq? a b))

(define (count nl)
     (node-list-count nl))

(define (node-list-count nl)
  (node-list-length (node-list-remove-duplicates nl)))

(define (node-list-remove-duplicates nl)
  (node-list-reduce nl
                (lambda (result snl)
                  (if (node-list-contains? result snl)
                      result
                      (node-list result snl)))
                (empty-node-list)))

(define (node-list-contains? nl snl)
  (node-list-reduce nl
                (lambda (result i)
                  (or result
                      (node-list=? snl i)))
                #f))

(define (string=? a b)
    (if (and (string? a) (string? b))
        (equal? a b)
        #f))
```

8 Page model set

```
;; -*- Scheme -*-
;;              Amended Pagemodel.dsl

;; Additional characteristics

;; Ruby character sequence cannot be extended over the Kanji
(declare-characteristic ruby-style "-//JIS//TR X 0010//JP" #t)

;; Patterns of line.
(declare-characteristic line-style "-//JIS//TR X 0010//JP" rule)

;; Ruby and emphasized mark don't increase line space.
(declare-characteristic layout-rule "-//JIS//TR X 0010//JP" #t)

;;
;; Page model

;; Default height of header/footer, for 2 lines.
(define *header-height* (* (cadddr *page-spec*) 2))
(define *footer-height* (* (cadddr *page-spec*) 2))

;; *page-region-y-offset* is distance from top left. Change into bottom left.
;; In addition to, region contains header/footer.

(define *pr-y-off*
  (- *paper-height* *page-region-y-offset*
     *page-region-height* *footer-height*))

;;;;;;; improvement (page-sequence);;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;(define-page-model standard-rect-page
;  (filling-direction 'top-to-bottom)
;  (width *paper-width*)
;  (height *paper-height*)
;  (region
;    (x-origin *page-region-x-offset*)
;    (y-origin *pr-y-off*)
;   (width *page-region-width*)
;   (height (+ *page-region-height* *header-height* *footer-height*))
```

```
;    (header
;      (height *header-height*)
;     (width *page-region-width*)
;      (generate (HEADER-CONTENT 'rect)))
;   (footer
;      (height *footer-height*)
;      (width *page-region-width*)
;      (generate (FOOTER-CONTENT 'rect)))
;   ))

;;;;;;; improvement (page-sequence);;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;(define *verso-pr-x-off*
;  (- *paper-width* *page-region-x-offset* *page-region-width*))

;;;;;;; improvement (page-sequence);;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;(define-page-model standard-verso-page
;  (filling-direction 'top-to-bottom)
;  (width *paper-width*)
;  (height *paper-height*)
;  (region
;   (x-origin *verso-pr-x-off*)
;   (y-origin *pr-y-off*)
;   (width *page-region-width*)
;   (height (+ *page-region-height* *header-height* *footer-height*))
;   (header
;    (height *header-height*)
;    (width *page-region-width*)
;    (generate (PAGE-HEADER 'verso)))
;   (footer
;    (height *footer-height*)
;    (width *page-region-width*)
;    (generate (PAGE-FOOTER 'verso)))
;   ))

;;
;; Specification of headline and page number.
;;
;;      *has-header-nonburu*
;;      *has-header-hasira*
```

```
;;      *has-footer-nonburu*
;;      *has-footer-hasira*
;;      *hasira-rect* (string)
;;      *hasira-verso* (string)
;;
;;   space for Headline and page number, Em.
;;

;;;;;;; improvement (page-sequence);;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;(define (PAGE-HEADER side)
;  (make paragraph
;        use: *header-footer-style* ;;*paragraph-style*
;        quadding: (if (eq side 'verso) 'end 'start)
;        (case side
;          (('rect)
;           (sosofo-append ((if *has-header-nonburu*
;                               (MAKE-NONBURU) #f)
;                           (literal "    ")
;                           (if *has-header-hasira*
;                               (literal *hasira-rect*) #f)
;           )))
;          (else
;           (sosofo-append ((if *has-header-hasira*
;               (literal *hasira-verso*) #f)
;           (if *has-header-nonburu*
;               (progn
;                (literal "    ")
;                (MAKE-NONBURU)) #f)
;           )))
;        )))

;;;;;;; improvement (page-sequence);;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;(define (PAGE-FOOTER side)
;  (make paragraph
;        use: *header-footer-style* ;;*paragraph-style*
;        space-before: *base-font-size* ;; 1 line space
;        quadding: (if (eq side 'verso) 'end 'start)
;        (case side
;          (('rect)
```

```
;          (sosofo-append ((if *has-footer-nonburu*
;                              (MAKE-NONBURU) #f)
;                          (literal "    ")
;                          (if *has-footer-hasira*
;                              (literal *hasira-rect*) #f)
;          )))
;         (else
;           (sosofo-append ((if *has-footer-hasira*
;               (literal *hasira-verso*) #f)
;           (if *has-footer-nonburu*
;               (progn
;                 (literal "    ")
;                 (MAKE-NONBURU)) #f)
;          )))
;       )))

(define MAKE-NONBURU
  ;;(literal "page ")
  (page-number-sosofo)  ;; Number only
)

;;
;; Set footnote
;;
(define (MAKE-FOOTNOTE-EXP num)  ;; Expression footnote
  (make-numbering num *footnote-number-desc*))

;;;;;;; improvement (included-container-area);;;;;;;;;;;;;;;;;;;;;;;;;;
;(define (FOOTNOTE)
;  (let ((footnote-exp (MAKE-FOOTNOTE-EXP (footnote-number 'page))))
;     ;; First, only footnote number every page.
;     ;; reference mark position of note is where (FOOTNOTE) is called .
;    (make sequence
;          (make included-container-area                to be revised
;                use: *footnote-style*
;                display?: #f
;                (literal footnote-exp))
;          (make paragraph
;                label: 'footnote
```

```
;               (literal footnote-exp)
;               (literal "    ")
;               (process-children)))))

;;;;;;; (included-container-area);;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;------Footnote number only-------;
(define (FOOTNOTE)
   (make superscript
      font-size: (* (inherited-font-size) *superscript-font-size-factor*)
      (literal (MAKE-FOOTNOTE-EXP (element-number (current-node))))))

;;;;;;; (included-container-area);;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;------Footnote contents----------;
(define (FOOTNOTE-CONTENTS)
   (with-mode FOOT-MODE
      (process-node-list (node-matching-list (parent (current-node)) "FN"))))

;;;;;;; (included-container-area);;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(mode FOOT-MODE
   (element FN
      (make paragraph
         (literal (MAKE-FOOTNOTE-EXP (element-number (current-node))))
         (process-children))))

;;;;;;; (included-container-area);;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;----Searches in the subgrove whose roots are each member of nl for element matching pattern.----;
(define (node-matching-list nl pattern)
   (let (
         (first (node-list-first  nl))
         (rest (node-list-rest  nl)))
      (if (node-list-empty? first)
         first
         (node-list
            (if (string=? pattern (gi first))
               first
               (node-list-rest first))
            (node-matching-list (children first) pattern)
            (node-matching-list rest pattern)))))
```

```
;;;;;;; improvement (included-container-area);;;;;;;;;;;;;;;;;;;;;;;;;;
;(define (FOOTNOTE-SEPARATOR)
;  (make rule
;        orientation: 'horizontal
;        line-thickness: 1pt
;        ))

;;
;; Set column
;;
;;;;;;; improvement (column-set-sequence);;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;(define *column-width* (* (cadr *page-spec*)(caddr *page-spec*)))
;(define *column-width+gap*
;  (let ((gap (cddddr *page-spec*)))
;    (if (null? gap)
;        *column-width*
;        (* (cadr *page-spec*)(+ (caddr *page-spec*) (car gap))))))

;;;;;;; improvement (column-set-sequence);;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;(define (filling-dir fld) ;; Filling direction is perpendicularity if
                          ;; how to set type is vertical.
;  (if (equal fld 'vertical)
;      'left-to-right
;      'top-to-bottom))

;;;;;;; improvement (column-set-sequence);;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;(define-column-set-model standard-one-column-model
;  (filling-direction (filling-dir *writing-dir*))
;  (column-subset
;   (column
;    (x-origin 0)
;    (width *column-width*)
;    (footnote-separator
;     (generate (FOOTNOTE-SEPARATOR))
;   (flow '(footnote footnote) )
;   ))
;))

;;;;;;; improvement (column-set-sequence);;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
;  (filling-direction (filling-dir *writing-dir*))
;  (column-subset
;   (column
;    (x-origin 0)
;    (width *column-width*))
;   (column
;    (x-origin *column-width+gap*)
;   (width *column-width*)
;   (footnote-separator
;   (generate (FOOTNOTE-SEPARATOR))
; (flow '(footnote footnote) )
;   ))
;))

;;;;;;; improvement (column-set-sequence);;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;(define-column-set-model standard-three-column-model
;  (filling-direction (filling-dir *writing-dir*))
;  (column-subset
;   (column
;    (x-origin 0)
;    (width *column-width*))
;   (column
;    (x-origin *column-width+gap*)
;    (width *column-width*))
;   (column
;    (x-origin (* 2 *column-width+gap*))
;    (width *column-width*)
;    (footnote-separator
;     (generate (FOOTNOTE-SEPARATOR))
;   (flow '(footnote footnote) )
;   ))
;))

;; Functions to generate top level flow object.
;;      Call with top level construction rule.
;;

;;;;;;; improvement (page-sequence, column-set-sequence, included-container-area);;;
;(define (STANDARD-PAGE-SEQUENCE)
```

```
;  (case *column-number*
;    ((1)
;     (make page-sequence                                    to be revised
;           initial-page-models: (standard-rect-page standard-verso-page)
;           repeat-page-models: (standard-rect-page standard-verso-page)
;           ;;content-map: '((footnote footnote))
;           (make column-set-sequence                        to be revised
;                 column-set-model: standard-one-column-model
;                 (process-children-trim))       ))
;    ((2)
;     (make page-sequence                                    to be revised
;           initial-page-models: (standard-rect-page standard-verso-page)
;           repeat-page-models: (standard-rect-page standard-verso-page)
;           ;;content-map: '((footnote footnote))
;           (make column-set-sequence                        to be revised
;                 column-set-model: standard-two-column-model
;                 (process-children-trim))       ))
;   ((3)
;    (make page-sequence                                     to be revised
;          initial-page-models: (standard-rect-page standard-verso-page)
;          repeat-page-models: (standard-rect-page standard-verso-page)
;           ;;content-map: '((footnote footnote))
;           (make column-set-sequence                       to be revised
;                 column-set-model: standard-three-column-model
;                 (process-children-trim))       ))
;    ))

;;;;;;; (page-sequence, column-set-sequence, included-container-area);;;;
(define (STANDARD-PAGE-SEQUENCE)
  (make simple-page-sequence
    font-family-name: *base-font-family*
    font-size: *base-font-size*
    line-spacing: (cadddr *page-spec*)
    left-header: (make sequence
                       font-size: (- *base-font-size* 1pt)
                       line-spacing: (cadddr *page-spec*)
                       font-posture: 'italic
                      (if-front-page
                       (empty-sosofo)
```

```
                (sosofo-append
                  (if *has-header-nonburu*
                    MAKE-NONBURU (empty-sosofo))
                  (if *has-header-hasira*
                    (literal *hasira-verso*) (empty-sosofo)))))
  right-header: (make sequence
                font-size: (- *base-font-size* 1pt)
                line-spacing: (cadddr *page-spec*)
                font-posture: 'italic
                (if-front-page
                  (sosofo-append
                    (if *has-header-hasira*
                      (literal *hasira-rect*) (empty-sosofo))
                    (if *has-header-nonburu*
                      MAKE-NONBURU (empty-sosofo)))
                  (empty-sosofo)))
  left-footer: (make sequence
                font-size: (- *base-font-size* 1pt)
                line-spacing: (cadddr *page-spec*)
                font-posture: 'italic
                (if-front-page
                  (empty-sosofo)
                  (sosofo-append
                    (if *has-footer-nonburu*
                      MAKE-NONBURU (empty-sosofo))
                    (if *has-footer-hasira*
                      (literal *hasira-verso*) (empty-sosofo)))))
  right-footer: (make sequence
                font-size: (- *base-font-size* 1pt)
                line-spacing: (cadddr *page-spec*)
                font-posture: 'italic
                (if-front-page
                  (sosofo-append
                    (if *has-footer-hasira*
                      (literal *hasira-rect*) (empty-sosofo))
                    (if *has-footer-nonburu*
                      MAKE-NONBURU (empty-sosofo)))
                  (empty-sosofo)))
  top-margin: *page-region-y-offset*
```

```
    bottom-margin: (- *paper-height* *page-region-height* *page-region-y-offset*)
    left-margin: *page-region-x-offset*
    right-margin: (- *paper-width* *page-region-width* *page-region-x-offset*)
    header-margin: (- *page-region-y-offset* *header-height*)
    footer-margin: *pr-y-off*
    page-width: *page-region-width*
    page-height: *page-region-height*
    quadding: 'justify
    page-n-columns: *column-number*
        (process-children-trim)
    ))

;;;;;;; (column-set-sequence);;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Characteristic definition for multi column-sets on simple-page-sequence.
(declare-characteristic page-n-columns
        "UNREGISTERED::James Clark//Characteristic::page-n-columns" 1)

;;;;;;; (page-sequence);;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Function definition to make verso and recto have difference sosofo.
(define if-front-page
  (external-procedure "UNREGISTERED::James Clark//Procedure::if-front-page"))
```

9 Flow object construction rules

```
;; -*- Scheme -*- amendment html.dsl

;; ======================== NON-PRINTING ELEMENTS ========================

;; Note that HEAD includes TITLE, ISINDEX, BASE, META, STYLE,
;;   SCRIPT, and LINK as possible children

;(default (empty-sosofo));; default element construction rule [171]

(element HEAD (empty-sosofo))
(element FORM (empty-sosofo))
(element APPLET (empty-sosofo))
(element PARAM (empty-sosofo))
(element TEXTFLOW (empty-sosofo))
(element MAP (empty-sosofo))
```

```
(element AREA (empty-sosofo))

;; ============================ TOP LEVEL ==============================

(element HTML
        (STANDARD-PAGE-SEQUENCE) ;; see pagemodel.dsl
)

(element BODY
        (process-children-trim))

;; ========================== BLOCK ELEMENTS ============================

;; ........................... Generic DIV ............................

(define (align-attr attr)
  (case attr
    (("LEFT") 'start)
    (("CENTER") 'center)
    (("RIGHT") 'end)
    (else 'justify)))

(element DIV
 (let ((align (align-attr (attribute-string "align"))))
   (make display-group
         quadding: align
         (process-children-trim))))

(element CENTER
 (make display-group
       quadding: 'center
       (process-children-trim)))


;; ........................... headings ..............................
(element H1 (TITLE-LARGE))   ;; see function.dsl
(element H2 (TITLE-MEDIUM))
(element H3 (TITLE-SMALL))
(element H4 (TITLE-SMALL))
```

```
(element H5 (TITLE-SMALL))
(element H6 (TITLE-SMALL))
;; ..... Paragraphs .....

(element P
 (make paragraph
       use: *fli-paragraph-style*
       quadding: (PQUAD)
       (process-children-trim)))

(element ADDRESS
  (make paragraph
        use: *paragraph-style*
        start-indent: *indent-step*
        (process-children-trim)))

(element BLOCKQUOTE
  (make paragraph
        start-indent: (+ (inherited-start-indent) *indent-step*)
        end-indent: (+ (inherited-end-indent) *indent-step*)
        (process-children-trim)))

(element PRE (MONO-SEQ))
(element XMP (MONO-SEQ))
(element LISTING (MONO-SEQ))
(element PLAINTEXT (MONO-SEQ))

(element BR
  (make display-group (empty-sosofo)))

;; ..... Lists .....

;;;    UL LI DIR MENU DL DT DD

(element OL (LIST-CONTAINER))
(element UL (LIST-CONTAINER))
(element DIR (LIST-CONTAINER))
(element MENU (LIST-CONTAINER))
```

```
(element (OL LI) (LIST-ELEMENT
        (make-numbering (child-number)
                (case (modulo (length (hierarchical-number-recursive "OL")) 4)
                  ((1) '(#f #f "(" last ")")) ;  (1)...
                  ((2) '(#f 'abc "(" last ")")) ; (a)...
                  ((3) '(#f 'roma "(" last ")")) ; (i)...
                  ((0) '(#f 'ABC "(" last ")")) ; (A)...
                ))))

(element (UL LI) (LIST-ELEMENT
        (case (modulo (length (hierarchical-number-recursive "UL")) 4)
          ((1) "-")
          ((2) " • ")
          ((3) "☆")
          ((0) "◎")
)))

(element (DIR LI) (LIST-ELEMENT " "))

(element (MENU LI) (LIST-ELEMENT " "))

(element DL (LIST-CONTAINER))

(element DT (make paragraph
        use: *paragraph-style*
        start-indent: (+ (inherited-start-indent)
                (* *indent-factor* *base-font-size*))
        first-line-start-indent: (- (* *indent-factor* *base-font-size*))
        (process-children)
        ))

(element DD (make paragraph
        use: *paragraph-style*
        start-indent: (+ (inherited-start-indent)
                (* *indent-factor* *base-font-size*))
        first-line-start-indent: 0pt
        (process-children)
        ))
```

```
;; ..... seq .....

(element B (BOLD-SEQ))
(element EM (BOLD-SEQ))
(element STRONG (BOLD-SEQ))
(element I (ITALIC-SEQ))
(element CITE (ITALIC-SEQ))
(element VAR (ITALIC-SEQ))
(element DFN (BOLD-ITALIC-SEQ))
(element A (BOLD-ITALIC-SEQ))

(element TT (MONO-SEQ))
(element CODE (MONO-SEQ))
(element KBD (MONO-SEQ))
(element SAMP (MONO-SEQ))

(element STRIKE (STRIKE-SEQ))
(element U (UNDERLINE))

;(element SUB (SUBSCRIPT))                    to be revised
(element SUB (SUBSCRIPT '()))
;(element SUP (SUPERSCRIPT '()))              to be revised
(element SUP (SUPERSCRIPT '()))

;; (element BIG )
;; (element SMALL )
;; (element FONT )

;; ============================== RULES ================================

(element HR
 (let ((align (attribute-string "ALIGN"))
       (noshade (attribute-string "NOSHADE"))
       (size (attribute-string "SIZE"))
       (width (attribute-string "WIDTH")))
  (make rule
        orientation: 'horizontal
        space-before: %block-sep%
        space-after: %block-sep%
```

```
        line-thickness: (if size (PARSEDUNIT size) 1pt)
        length: (if width (PARSEDUNIT width) %body-width%)
        display-alignment:
          (case align
                (("LEFT") 'start)
                (("CENTER") 'center)
                (("RIGHT") 'end)
                (else 'end)))))


;; ============================== GRAPHICS ================================

;; Note that DSSSL does not currently support text flowed around an
;;   object, so the action of the ALIGN attribute is merely to shift the
;;   image to the left or right.  An extension to add runarounds to DSSSL
;;   has been proposed and should be incorporated here when it becomes
;;   final.

(element IMG
  (make external-graphic
        entity-system-id: (attribute-string "src")
        display?: #t
        space-before: 1em
        space-after: 1em
        display-alignment:
          (case (attribute-string "align")
                (("LEFT") 'start)
                (("RIGHT") 'end)
                (else 'center))))


;; ============================== TABLES ================================

(element TABLE
;; number-of-columns is for future use
  (let ((number-of-columns
         (node-list-reduce (node-list-rest (children (current-node)))
                           (lambda (cols nd)
                             (max cols
```

```
                        (node-list-length (children nd))))
                0)))
  (make display-group
        space-before: %block-sep%
        space-after: %block-sep%
        start-indent: %body-start-indent%
;; for debugging:
;;      (make paragraph
;;            (literal
;;             (string-append
;;              "Number of columns: "
;;              (number->string number-of-columns))))
        (with-mode table-caption-mode (process-first-descendant "CAPTION"))
        (make table
              (process-children)))))

(mode table-caption-mode
  (element CAPTION
           (make paragraph
                 use: para-style
                 font-weight: 'bold
                 space-before: %block-sep%
                 space-after: %para-sep%
                 start-indent: (inherited-start-indent);
                 (literal
                  (string-append
                   "Table "
                   (format-number
                    (element-number) "1") ". "))
                 (process-children-trim))))

(element CAPTION (empty-sosofo)) ; don't show caption inside the table

(element TR
  (make table-row
        (process-children-trim)))

(element TH
  (make table-cell
```

```
        ;n-rows-spanned: (string->number (attribute-string "COLSPAN"))
        (make paragraph
              font-weight: 'bold
              space-before: 0.25em
              space-after: 0.25em
              start-indent: 0.25em
              end-indent: 0.25em
              quadding: 'start
              (process-children-trim))))

(element TD
  (make table-cell
        ;n-rows-spanned: (string->number (attribute-string "COLSPAN"))
        (make paragraph
              space-before: 0.25em
              space-after: 0.25em
              start-indent: 0.25em
              end-indent: 0.25em
              quadding: 'start
              (process-children-trim))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(define (MONO-SEQ)
  (make sequence
      (process-children)))
(define %para-sep% (/ *base-font-size* 2.0))
(define %block-sep% (* %para-sep% 2.0))
(define %body-width% *page-region-width*)
(define (PQUAD)
    (case (attribute-string "align")
        (("LEFT") 'start)
        (("CENTER") 'center)
        (("RIGHT") 'end)
        (else (inherited-quadding))))

;a definition of style
(define para-style
  (style
   font-size: *base-font-size*
```

```
   line-spacing: (* *base-font-size* 1.1)))

;a definition of unit
(define-unit em *base-font-size*)
(define-unit pi (/ 1in 6))
(define-unit px (/ 1in 96))
(define-unit mm .001m)
(define-unit cm .01m)

;a definition of functions
(define (node-list-reduce nl combine init)
 (if (node-list-empty? nl)
      init
      (node-list-reduce (node-list-rest nl)
           combine
           (combine init (node-list-first nl)))))

(define upperalpha '(A))
;  (list #\A #\B #\C #\D #\E #\F #\G #\H #\I #\J #\K #\L #\M
;        #\N #\O #\P #\Q #\R #\S #\T #\U #\V #\W #\X #\Y #\Z))

(define loweralpha '(a))
;  (list #\a #\b #\c #\d #\e #\f #\g #\h #\i #\j #\k #\l #\m
;        #\n #\o #\p #\q #\r #\s #\t #\u #\v #\w #\x #\y #\z))

(define (EQUIVLOWER c a1 a2)
  (cond ((null? a1) '())
        ((char=? c (car a1)) (car a2))
        ((char=? c (car a2)) c)
        (else (EQUIVLOWER c (cdr a1) (cdr a2)))))

(define (char-downcase c)
  (EQUIVLOWER c upperalpha loweralpha))

(define (ISALPHA? c)
  (if (or (member c upperalpha) (member c loweralpha)) #t #f))

(define (LOCASE slist)
  (if (null? slist)
```

```
      '()
      (cons (char-downcase (car slist)) (LOCASE (cdr slist)))))

(define (STR2LIST s)
  (let ((start 0)
        (len (string-length s)))
    (let loop ((i start) (l len))
         (if (= i len)
             '()
             (cons (string-ref s i)(loop (+ i 1) l))))))

(define (LIST2STR x)
  (apply string x))

(define (STRING-DOWNCASE s)
  (LIST2STR (LOCASE (STR2LIST s))))

(define (UNAME-START-INDEX u last)
  (let ((c (string-ref u last)))
    (if (ISALPHA? c)
        (if (= last 0)
            0
            (UNAME-START-INDEX u (- last 1)))
        (+ last 1))))

(define (PARSEDUNIT u)
  (if (string? u)
    (let ((strlen (string-length u)))
      (if (> strlen 2)
          (let ((u-s-i (UNAME-START-INDEX u (- strlen 1))))
            (if (= u-s-i 0)
                1pi
                (if (= u-s-i strlen)
                    (* (string->number u) 1px)
                    (let* ((unum (string->number
                                  (substring u 0 u-s-i)))
                           (uname (STRING-DOWNCASE
                                   (substring u u-s-i strlen))))
                      (case uname
```

```
                    (("mm") (* unum 1mm))
                    (("cm") (* unum 1cm))
                    (("in") (* unum 1in))
                    (("pi") (* unum 1pi))
                    (("pc") (* unum 1pi))
                    (("pt") (* unum 1pt))
                    (("px") (* unum 1px))
                    (("barleycorn") (* unum 2pi))
                    (else
                      (cond
                        ((number? unum)
                         (* unum 1px))
                        ((number? (string->number u))
                         (* (string->number u) 1px))
                            (else u))))))))
        (if (number? (string->number u))
            (* (string->number u) 1px)
            1pi)))
  1pi))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(element YOMI (YOMI))
(element FN (FOOTNOTE))
(element FN-CONTENTS (FOOTNOTE-CONTENTS))
```