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))
```