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)

;