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))
```