〔公開〕

TR－C－0052

Ｓｙｍｂｏｌｉｃｓ用　ｄｖｉファイル　プレビュアーシステム

林　潔　KIYOSHI HAYASHI
佐藤　隆　TAKASHI SATO

１９９０．　８．　１．

ＡＴＲ通信システム研究所

Symbolics 用　ＤＶＩファイルプレビュアーシステム

１．概　要

シンボリックスＬＩＳＰマシン上でＴＥＸの出力イメージをプレビューするためのシステムであり、シンボリックスのＳＹＳＴＥＭ機能を用いて実現している。

２．利用方法

2.1 システムのロード

シンボリックスのリスプリスナーのコマンド入力で、

```
Command: Load System dvi
```

と入力することによってシステムがロードされ仕様可能状態になる。

2.2 システムの起動

プレビュアーウインドウは、「 Select 」＋「ｖ」のキー入力により起動され図１の枠組みのような画面になる。
ここで、受付られるコマンドは、左上に用意された３種類で、それぞれマウスクリックすることによって起動できる。

2.3 コマンド説明

2.3.1 DVI Previewer

図１のようなポップアップメニューによりｄｖｉファイルのパス名等設定でき、Exit　部をマウスクリックすることでプレビュー開始となる。

2.3.2 Font Utilities

ポップアップメニューにより現在自分のマシンにロードされているフォントのリスト出力とフォントの表示が可能である。
なお、自分のマシンにないフォントはディフォルトで cs01:/usr/lib/tex/fonts/ ディレクトリ（dvi::*dvi-pixel-host* に設定されている）から取り込まれ、自分のマシンの local:>dvi>tex-fonts> ディレクトリ（dvi::*dvi-pixel-path* に設定されている）配下にコピーされる。
（但し、フォントのコピーが無い場合は表示速度が極端に遅い）

2.3.3 Others

ポップアップメニューにより図１に示す様に、メニューのリセットとガーベージコレクタ起動の２種類のコマンドがマウスから起動可能となる。

2.4 プレビューアー画面のスクロール

プレビューされた画面は左側のスクロールバーをマウス操作することにより、スクーロールアップ／ダウンが可能です。

3．保　守

シンボリックスのシステム機能を用いて実現しているためシステムのディストリビューション、リストア，エディット及び、パッチ機能等はシンボリックスマニュアル参照。

4．諸　元

・リリースバージョン　　3.0　　Genera 7.2 対応

・ソースファイル

clm01:>local>system>dvi-previewer>define-system.lisp
〃　　>control-pane.lisp
〃　　>common-function.lisp
〃　　>font-utilities.lisp
〃　　>show-previewer.lisp

5．付　録

ｄｖｉプレビューアー　ソース　ファイル一式

以　上

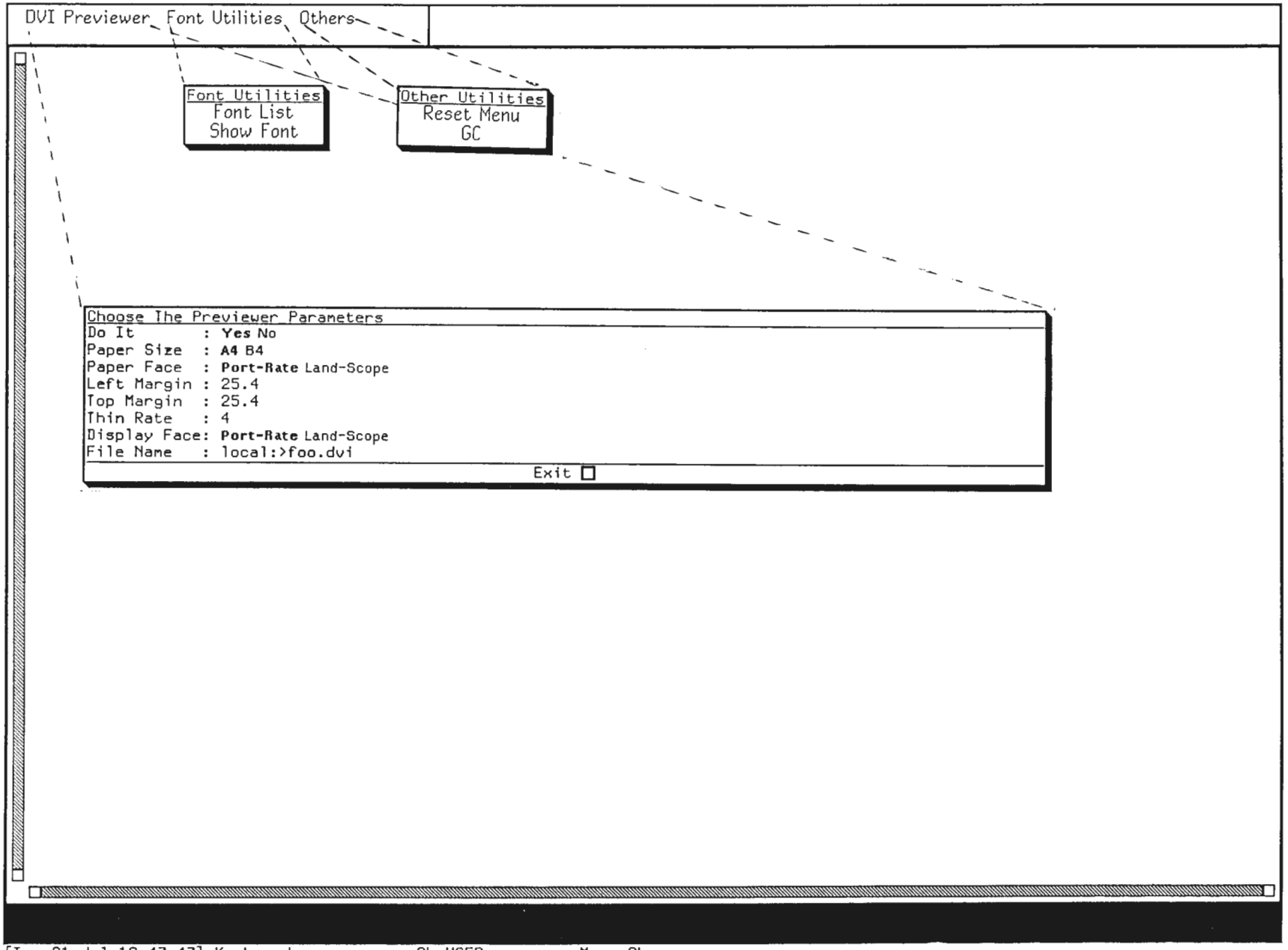
DVI Previewer Font Utilities Others
Font Utilities
Font List
Show Font
Other Utilities
Reset Menu
GC
Choose The Previewer Parameters
Do It : Yes No
Paper Size : A4 B4
Paper Face : Port-Rate Land-Scope
Left Margin : 25.4
Top Margin : 25.4
Thin Rate : 4
Display Face: Port-Rate Land-Scope
File Name : local:>foo.dvi
Exit
[Tue 31 Jul 12:47:47] Keyboard
CL USER:
Menu Choose

図1. プルダウン画面

付　録：　ソースリスト

CLM01:>local>system>dvi-previewer>define-system.lisp.1

For: Kiyoshi Hayashi
Printed on: comprinter
Number of copies: 1
Data created at: 3/01/90 17:07:14
Queued at: 7/31/90 13:01:11

```
;;; -*- Mode: LISP; Package: USER; Syntax: Common-lisp; Base: 10 -*-
;;;
;;;    DVI File Previewer System
;;;

(Defpackage DVI
  (:nicknames DVI-Previewer)
  (:use SCL)
  )


;;; Define System

(defsystem DVI
    (:pretty-name "DVI File Previewer System"
     :default-pathname "DVI:previewer;"
     :default-package DVI
     )
  (:serial
    "control-pane"
    "common-functions"
    "font-utilities"
    "show-previewer"
    ))


;;;
;;;  end of file
;;;
```

CLM01:>local>system>dvi-previewer>control-pane.lisp.5

For: Kiyoshi Hayashi
Printed on: comprinter
Number of copies: 1
Data created at: 6/22/90 18:26:34
Queued at: 7/31/90 13:01:23

```
;;; -*- Mode: LISP; Syntax: Common-lisp; Base: 10; Package: DVI -*-

(defvar *dvi-sheet-pane* nil)
(defvar *dvi-info-pane* nil)
(defvar *dvi-menu-font-util-1*)
(defvar *dvi-menu-font-util-2*)
(defvar *dvi-menu-other-util-1*)
(defvar *dvi-file-name*    "local:>foo.dvi")
(defvar *dvi-paper-size*   :a4)
(defvar *dvi-paper-face*   :port-rate)
(defvar *dvi-left-margin*  25.4)
(defvar *dvi-top-margin*   25.4)
(defvar *dvi-thin-rate*    4)
(defvar *dvi-display-face* :port-rate)

(defun reset-menu ()
  (setf *dvi-menu-font-util-1*
        (tv:make-window
          'tv:momentary-menu
          ':label "Font Utilities"
          ':item-list '(("Font List"
                         :funcall show-font-list
                         :documentation "フォント一覧の表示")
                        ("Show Font"
                         :funcall show-font-raster
                         :documentation "フォントの表示"))))
  (setf *dvi-menu-font-util-2*
        (tv:make-window
          'tv:momentary-menu
          ':label "Selection"))
  (setf *dvi-menu-other-util-1*
        (tv:make-window
          'tv:momentary-menu
          ':label "Other Utilities"
          ':item-list '(("Reset Menu"
                         :funcall reset-menu
                         :documentation "メニューのリセット")
                        ("GC"
                         :funcall gc-immediately
                         :documentation "ガーベジ・コレクションの開始"))))
  )

(dw:define-program-framework previewer
  :select-key #\v
  :command-definer t
  :command-table (:kbd-accelerator-p t
                  :inherit-from nil)
  :top-level (dw:default-command-top-level :echo-stream ignore)
  :state-variables nil
  :panes
  ((sheet :display
          :redisplay-after-commands nil
          :redisplay-function 'show-sheet
          :incremental-redisplay nil
          :typeout-window t
          :margin-components 'dw:((margin-ragged-borders)
                                  (margin-scroll-bar :history-noun "sheet")
                                  (margin-scroll-bar :history-noun "sheet"
                                                     :margin :bottom)
                                  (margin-white-borders :thickness 2)))
   (info :display
         :redisplay-after-commands nil
         :redisplay-function 'show-info)
   (menu  :command-menu :menu-level :top-level :columns 3))
  :configurations
  '((dw::main
      (:layout
        (dw::main :column row-1 sheet)
        (row-1 :row menu info))
      (:sizes
        (dw::main (sheet 49 :lines) :then (row-1 :even))
        (row-1 (info 0.67) :then (menu :even))))))

(defmethod (show-sheet previewer) (sheet-pane)
  (setf *dvi-sheet-pane* sheet-pane)
  (send *dvi-sheet-pane* :refresh)
  (reset-menu))
```

```
(defmethod (show-info previewer) (info-pane)
  (setf *dvi-info-pane* info-pane)
  (setf *dvi-font-infout* *dvi-info-pane*)
  (setf *dvi-prev-infout* *dvi-info-pane*)
  (send *dvi-info-pane* :refresh)
  (send *dvi-info-pane* :home-cursor))

(define-previewer-command (com-font-utilities :menu-accelerator "Font Utilities") ()
     (send *dvi-menu-font-util-1* ':choose))

(defvar *tmp-do-prev*)
(defvar *tmp-pp-size*)
(defvar *tmp-pp-face*)
(defvar *tmp-lf-marg*)
(defvar *tmp-tp-marg*)
(defvar *tmp-th-rate*)
(defvar *tmp-ds-face*)
(defvar *tmp-fl-name*)

(define-previewer-command (com-dvi-previewer :menu-accelerator "DVI Previewer") ()
   (send *dvi-sheet-pane* :clear-history)
   (send *dvi-sheet-pane* :refresh)
   (send *dvi-sheet-pane* :home-cursor)
   (setf *tmp-do-prev* :yes)
   (setf *tmp-pp-size* *dvi-paper-size*)
   (setf *tmp-pp-face* *dvi-paper-face*)
   (setf *tmp-lf-marg* *dvi-left-margin*)
   (setf *tmp-tp-marg* *dvi-top-margin*)
   (setf *tmp-th-rate* *dvi-thin-rate*)
   (setf *tmp-ds-face* *dvi-display-face*)
   (setf *tmp-fl-name* *dvi-file-name*)
   (tv:choose-variable-values
     '((*tmp-do-prev* "Do It       " :assoc (("Yes" . :yes) ("No" . :no)))
       (*tmp-pp-size* "Paper Size  " :assoc (("A4" . :a4) ("B4" . :b4)))
       (*tmp-pp-face* "Paper Face  "
                  :assoc (("Port-Rate" . :port-rate) ("Land-Scope" . :land-scope)))
       (*tmp-lf-marg* "Left Margin " :number)
       (*tmp-tp-marg* "Top Margin  " :number)
       (*tmp-th-rate* "Thin Rate   " :number)
       (*tmp-ds-face* "Display Face"
                  :assoc (("Port-Rate" . :port-rate) ("Land-Scope" . :land-scope)))
       (*tmp-fl-name* "File Name   " :string))
     ':label "Choose The Previewer Parameters"
     ':width 100)
   (if (eq :yes *tmp-do-prev*)
       (let ()
         (setf *dvi-paper-size* *tmp-pp-size*)
         (setf *dvi-paper-face* *tmp-pp-face*)
         (setf *dvi-left-margin* *tmp-lf-marg*)
         (setf *dvi-top-margin* *tmp-tp-marg*)
         (setf *dvi-thin-rate* *tmp-th-rate*)
         (setf *dvi-display-face* *tmp-ds-face*)
         (setf *dvi-file-name* *tmp-fl-name*)
         (dvi-previewer *dvi-file-name* *dvi-sheet-pane*)))
   ;; (format *dvi-info-pane* "~%Done DVI PREVIEWER.")
   )

(define-previewer-command (com-other-utilities :menu-accelerator "Others") ()
     (send *dvi-menu-other-util-1* ':choose))

(defun show-font-list ()
  (send *dvi-sheet-pane* :clear-history)
  (send *dvi-sheet-pane* :refresh)
  (send *dvi-sheet-pane* :home-cursor)
  (dolist (font *dvi-font-stack*)
    (format *dvi-sheet-pane* "~%Name=~S" (fnts-name font)))
  ;; (format *dvi-info-pane* "~%Done SHOW FONT LIST.")
  )

(defun show-font-raster ()
  ;; (format *dvi-info-pane* "~%BeginSFR")
  (do ((fonts *dvi-font-stack* (cdr fonts))
       (names nil)
       (name))
      ((null fonts)
       (send *dvi-menu-font-util-2* ':set-item-list names)
       (setf name (send *dvi-menu-font-util-2* ':choose))
       ;; (format *dvi-info-pane* " ~S" name)
```

```
      (if name
          (let ()
            (send *dvi-sheet-pane* :clear-history)
            (send *dvi-sheet-pane* :refresh)
            (send *dvi-sheet-pane* :home-cursor)
            (show-font 100 100 (find-font name) *dvi-sheet-pane*))))
    (push (fnts-name (car fonts)) names))
  ;; (format *dvi-info-pane* "~%Done SHOW FONT RASTER.")
  )
```

CLM01:>local>system>dvi-previewer>common-functions.lisp.3

For: Kiyoshi Hayashi
Printed on: comprinter
Number of copies: 1
Data created at: 7/31/90 12:55:45
Queued at: 7/31/90 13:01:51

```
;;; -*- Mode: LISP; Syntax: Common-lisp; Base: 10; Package: DVI -*-
;;;
;;; (read-unsigned-bytes stream count)
;;; (read-signed-bytes stream count)
;;;
;;; stream file から count で指定された data を入力し、整数値に変換し、return value として返す。
;;; 入力 data は、上位の桁から下位の桁へと順に作成されているものとする。
;;; read-unsigned-bytes は符号無し、read-signed-bytes は符号付きで変換する。
;;; stream file は、opne 時に、以下の option が指定されていなければならない。
;;;     :characters nil :byte-size 8

(defun read-unsigned-bytes (stream count)
  (do ((k count (1- k))
       (cha)
       (val 0))
      ((<= k 0) val)
    (setf cha (read-byte stream nil nil))
    (if (null cha)
        (return-from read-unsigned-bytes nil))
    (setf val (+ (* 256 val) cha))))

(defun read-signed-bytes (stream count)
  (let ((cha (read-byte stream nil nil))
        (val))
    (if (null cha)
        (return-from read-signed-bytes nil))
    (if (> 128 cha)
        (setf val cha)
        (setf val (- cha 256)))
    (do ((k (1- count) (1- k)))
        ((<= k 0) val)
      (setf cha (read-byte stream nil nil))
      (if (null cha)
          (return-from read-signed-bytes nil))
      (setf val (+ (* 256 val) cha)))))
```

CLM01:>local>system>dvi-previewer>font-utilities.lisp.5

For: Kiyoshi Hayashi
Printed on: comprinter
Number of copies: 1
Data created at: 7/31/90 12:54:44
Queued at: 7/31/90 13:02:00

```
;;; -*- Mode: LISP; Syntax: Common-lisp; Base: 10; Package: DVI -*-

(defstruct fnts
  (name)                                    ; Font Name
  (scal)                                    ; Font Scale
  (size)                                    ; Design Size
  (magn)                                    ; Magnification
  (leng)                                    ; Font Count
  (char)                                    ; Characters Data
  (dpos)                                    ; Pixel Directory's Position
  (buff))                                   ; File Buffer (word)

(defstruct pxls
  (widt)                                    ; Pixel width [dot]
  (heig)                                    ; Pixel height [dot]
  (xoff)                                    ; X Offset [dot]
  (yoff)                                    ; Y Offset [dot]
  (tfmw)                                    ; TFM width
  (rast))                                   ; Raster Data

(defmacro dvi-dot (dvi)
  `(round (/ (* ,dvi num *dvi-resolution*) den 254000 *dvi-thin-rate*)))

(defmacro n-of-m (n m) `(let ((l (mod ,n ,m))) (if (zerop l) ,m l)))

;(defvar *dvi-thin-rate* 4)                 ; font の間引き率
(defvar *dvi-pixel-path* "local:>dvi>tex-fonts>")
(setq *dvi-pixel-host* "cs01:/usr/lib/tex/fonts/")
(defvar *dvi-resolution* 300)               ; Screeen Resolution [DPI]
(defvar *dvi-font-stack* nil)
(defvar *dvi-font-infout* *standard-output*)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;;; SHOW-FONT
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
(defun show-font (max-w max-h font-data &optional (window *standard-output*))
  (let ((pixels (fnts-char font-data))
        (max-w1) (max-h1)
        (num 25400000)                      ; Numerator (TeX Standard Value)
        (den 473628672)                     ; Denominator (TeX Standard Value)
        (nw) (nh)
        (x 0) (y 0))
    ;; (format *dvi-font-infout* "~%ShowFont W=~S, H=~S" max-w max-h)
    ;; (format *dvi-font-infout* "~% ~S" font-data)
    (setf max-w1 (1+ max-w))
    (setf max-h1 (1+ max-h))
    (setf nw 8)
    (setf nh (ceiling (/ (fnts-leng font-data) nw)))
    (dotimes (i (1+ nw))
      (graphics:draw-line (* i max-w1) 0
                          (* i max-w1) (1+ (* max-h1 nh))
                          :stream window
                          :dashed t :dash-spacing 4 :dash-length 1))
    (dotimes (i (1+ nh))
      (graphics:draw-line 0 (* i max-h1)
                          (1+ (* nw max-w1)) (* i max-h1)
                          :stream window
                          :dashed t :dash-spacing 4 :dash-length 1))
    (dotimes (k (fnts-leng font-data))
      (if (zerop (mod k nw))
          (let ()
            (setf x 1)
            (incf y max-h1))
          (incf x max-w1))
      (if (write-font window x y k font-data)
          (graphics:draw-line (+ x (dvi-dot (pxls-tfmw (aref pixels k)))) y
                              (+ x (dvi-dot (pxls-tfmw (aref pixels k)))) (- y max-h)
                              :stream window
                              :dashed t :dash-spacing 4 :dash-length 1)
          ))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;;; WRITE-FONT
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
```

```
(defun int-16bit (x) (if (< #x7FFF x) (- x #x10000) x))

(defun write-font (window x-pos y-pos pixel-no font)
  (if (eq t (aref (fnts-char font) pixel-no))
      (let ((pixel (make-pxls))
            (thin (list 0))
            (pxlbuf (fnts-buff font))           ; Pixel Data (array:from *.*pxl)
            (pxldir (+ (fnts-dpos font) (* 4 pixel-no)))
                                                ; Beginning Position of Pixel Directory
            (pxlpos)                            ; Beginning Position of Pixel Data
            (pxl (make-list *dvi-thin-rate* :initial-element 0))        ; 1 文字分の dot patter
n
            (pxlw)                              ; Pixel Width
            (pxlh)                              ; Pixel Height
            (shift)
            (dots))                             ; 1 column 分の dot pattern
        (setf pxlw (floor (/ (aref pxlbuf pxldir) #x10000)))
        (setf pxlh (mod (aref pxlbuf pxldir) #x10000))
        (if (zerop pxlw)
            (let ()
              (setf (aref (fnts-char font) pixel-no) nil)
              (return-from write-font nil)))
        (setf pxlpos (1- (aref pxlbuf (+ pxldir 2))))
        ;;
        ;; (format *dvi-font-infout* "~%~D: W=~D, H=~D, Offs(~D,~D), TFM-W=~D "
        ;; pixel-no pxlw pxlh
        ;; (pxls-xoff pixel) (pxls-yoff pixel) (pxls-tfmw pixel))
        ;;
        ;; 'pxl-list' から 1 文字分の dot pattern を取り出す。
        ;;
        (setf shift (expt 2 (mod (- 32 pxlw) 32)))
        (dotimes (l pxlh)
          (case (ceiling (/ pxlw 32))
            (1
              (setf dots (aref pxlbuf pxlpos))
              (incf pxlpos))
            (2
              (setf dots (+ (* #x100000000 (aref pxlbuf pxlpos)) (aref pxlbuf (1+ pxlpos))))
              (incf pxlpos 2))
            (otherwise
              (setf dots 0)
              (dotimes (m (ceiling (/ pxlw 32)))
                (setf dots (+ (* #x100000000 dots) (aref pxlbuf pxlpos)))
                (incf pxlpos))))
          (nconc pxl (list (/ dots shift))))
        (setf (pxls-rast pixel) pxl)
        ;; (format *dvi-font-infout* "~%PXL ~S ~S" (length pxl) pxl)
        (do ((l 0 (+ l *dvi-thin-rate*))
             (x (nthcdr (n-of-m pxlh *dvi-thin-rate*) pxl)
                (nthcdr *dvi-thin-rate* x)))
            ((>= l pxlh))
          (nconc thin (list (thin-4-int (car x) (cadr x) (caddr x) (cadddr x) pxlw))))
        (setf (pxls-widt pixel) (ceiling (/ pxlw *dvi-thin-rate*)))
        (setf (pxls-heig pixel) (ceiling (/ pxlh *dvi-thin-rate*)))
        (setf (pxls-xoff pixel)
              (round (/ (int-16bit (floor (/ (aref pxlbuf (1+ pxldir)) #x10000)))
                        *dvi-thin-rate*)))
        (setf (pxls-yoff pixel)
              (round (/ (int-16bit (mod (aref pxlbuf (1+ pxldir)) #x10000))
                        *dvi-thin-rate*)))
        (setf (pxls-tfmw pixel)
              (floor (/ (* (aref pxlbuf (+ pxldir 3)) (fnts-scal font)) #x100000)))
        ;; (format *dvi-font-infout* "~%H=~S, W=~S, R=~S~%"
        ;;       (pxls-heig pixel) (pxls-widt pixel) (length (cdr thin)))
        ;; (dolist (l (cdr thin)) (put-bit l (pxls-widt pixel) "." "X" *dvi-font-infout*))
        (setf (pxls-rast pixel) (tv:make-binary-gray
                                  (pxls-heig pixel) (pxls-widt pixel) (cdr thin)))
        (setf (aref (fnts-char font) pixel-no) pixel)
        ))
  (if (null (aref (fnts-char font) pixel-no))
      (return-from write-font nil))
  (let* ((pixel (aref (fnts-char font) pixel-no))
         (x (- x-pos (pxls-xoff pixel)))
         (y (- y-pos (pxls-yoff pixel))))
    ;; (format *dvi-font-infout* "~%WF ~S ~S ~S" x y (array-dimensions (pxls-rast pixel)))
    (graphics:draw-pattern x y (pxls-rast pixel) :stream window
                           :right (+ x (pxls-widt pixel))
                           :bottom (+ y (pxls-heig pixel)))))
```

```
  t)

(defun thin-2-int (x1 x2 bit)
  (setf (ldb (byte 1 bit) x1) 1)
  (do ((i 0 (+ i 2))
       (j 0 (1+ j))
       (v 0))
      ((>= i bit) v)
    (if (< 1 (+ (ldb (byte 1 i) x1) (ldb (byte 1 (1+ i)) x1)
                (ldb (byte 1 i) x2) (ldb (byte 1 (1+ i)) x2)))
        (setf (ldb (byte 1 j) v) 1))))

(defun thin-3-int (x1 x2 x3 bit)
  (setf (ldb (byte 3 bit) x1) 1)
  (setf (ldb (byte 3 bit) x3) 1)
  (do ((i 0 (+ i 3))
       (j 0 (1+ j))
       (v 0))
      ((>= i bit) v)
    (if (< 2 (+ (ldb (byte 1 i) x1) (ldb (byte 1 (1+ i)) x1) (ldb (byte 1 (+ i 2)) x1)
                (ldb (byte 1 i) x2) (ldb (byte 1 (1+ i)) x2) (ldb (byte 1 (+ i 2)) x2)
                (ldb (byte 1 i) x3) (ldb (byte 1 (1+ i)) x3) (ldb (byte 1 (+ i 2)) x3)
                ))
        (setf (ldb (byte 1 j) v) 1))))

(defun thin-4-int (x1 x2 x3 x4 bit)
  (setf (ldb (byte 3 bit) x1) 1)
  (setf (ldb (byte 3 bit) x3) 1)
  (do ((i 0 (+ i 4))
       (j 0 (1+ j))
       (v 0))
      ((>= i bit) v)
    (if (< 3 (+ (ldb (byte 1 i) x1) (ldb (byte 1 (1+ i)) x1) (ldb (byte 1 (+ i 2)) x1) (ldb (
byte 1 (+ i 3)) x1)
                (ldb (byte 1 i) x2) (ldb (byte 1 (1+ i)) x2) (ldb (byte 1 (+ i 2)) x2) (ldb (
byte 1 (+ i 3)) x2)
                (ldb (byte 1 i) x3) (ldb (byte 1 (1+ i)) x3) (ldb (byte 1 (+ i 2)) x3) (ldb (
byte 1 (+ i 3)) x3)
                (ldb (byte 1 i) x4) (ldb (byte 1 (1+ i)) x4) (ldb (byte 1 (+ i 2)) x4) (ldb (
byte 1 (+ i 3)) x4)))
        (setf (ldb (byte 1 j) v) 1))))

(defun put-bit (x bit p0 p1 &optional (window t))
  (dotimes (i bit)
    (if (zerop (ldb (byte 1 (- bit i 1)) x))
        (format window "~A" p0)
        (format window "~A" p1)))
  (format window "~%"))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;;; FIND-FONT
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
(defun find-font (name)
  (dolist (font *dvi-font-stack*)
    (if (string= (fnts-name font) name)
        (return-from find-font font)))
  nil)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;;; MAKE-FONT
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
(defun make-font (new-font scale)
  (let ((font (find-font new-font)))
    (if (null font)
        (let ()
          (setf font (read-font new-font scale))
          (push font *dvi-font-stack*)))
    font))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;;; READ-FONT
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
```

```
(defun read-font (new-font scale)
  ;; (format t "~%Begin READ-FONT ")
  (let ((file-size)                              ; File の byte 数
        (file-w-size)
        (font-list (make-fnts)))                 ; Font 情報用の list
    ;; (let ((time1 (zl:time)) (time2))
      ;;
      ;; PIXEL FORMAT の data を最後まで入力する。
      ;; (format *dvi-font-infout* "~%Begin READ-PIXCEL ~S" new-font)
      ;;
      (setf (fnts-name font-list) new-font)
      (setf (fnts-scal font-list) scale)
      (let ((file-name)                          ; Font file 名
            (pixel-id))
        (setf file-name (concatenate 'string *dvi-pixel-path* new-font))
        (if (null (open file-name :direction :probe))
            (let ((from-file-name))
              ;; (format *dvi-font-infout* "~%No file ~S" file-name)
              (setf from-file-name (concatenate 'string *dvi-pixel-host* new-font))
              (si:copyf from-file-name file-name :byte-size 8 :characters nil)
              ))
        (with-open-file (pxl-file file-name :direction :input :characters nil :byte-size 8)
          ;;
          ;; File の先頭にある ID 番号を check する。
          ;;
          (setf pixel-id (read-unsigned-bytes pxl-file 4))
          (case pixel-id
            (1001 t)                             ; 4 bytes boundary
            (1002                                ; 1 bytes boundary
              (format *dvi-font-infout*
                      "~%ID番号(~D)のファイル~Sは入力できません。" pixel-id file-name)
              (return-from read-font nil))
            (t
              (format *dvi-font-infout*
                      "~%ファイル~SのID番号(~D)は理解できません。" file-name pixel-id)
              (return-from read-font nil)))
          ;;
          ;; File の size を計算し、check する。
          ;;
          (setf file-size (file-length pxl-file))
          (if (not (zerop (mod file-size 4)))
              (let ()
                (format *dvi-font-infout*
                        "~%ファイル~Sのサイズ(~D)が不適切です。" file-name file-size)
                (return-from read-font nil)))
          ;;
          ;; (format *dvi-font-infout* "~%File Name: ~D" file-name)
          ;; (format *dvi-font-infout* "~%File Size: ~D" file-size)
          ;;
          ;; Buffer を用意し、raster と PXL-DIR を入力する。
          ;;
          (setf file-w-size (/ file-size 4))
          (setf (fnts-buff font-list) (make-array (- file-w-size 6)))
          (dotimes (i (- file-w-size 6))
            (setf (aref (fnts-buff font-list) i) (read-unsigned-bytes pxl-file 4)))
          ;;
          ;; 残りの情報を入力する。
          ;;
          (read-unsigned-bytes pxl-file 4)       ; Check Digit は無視する。
          (setf (fnts-magn font-list) (read-signed-bytes pxl-file 4))
          (setf (fnts-size font-list) (read-signed-bytes pxl-file 4))
          (setf (fnts-dpos font-list) (1- (read-signed-bytes pxl-file 4)))
          ;; 最後の ID 番号は無視する。
          ;; (format t "~%Done. Get Pixel ")
          ))
      (setf (fnts-leng font-list) (/ (- file-w-size 6 (fnts-dpos font-list)) 4))
      (setf (fnts-char font-list) (make-array (fnts-leng font-list) :initial-element t))
      ;;
      ;; (format *dvi-font-infout* "~%Font Name:     ~S" (fnts-name font-list))
      ;; (format *dvi-font-infout* "~%Design Size:   ~S" (fnts-size font-list))
      ;; (format *dvi-font-infout* "~%Magnification: ~S" (fnts-magn font-list))
      ;; (format *dvi-font-infout* "~%Scale:         ~S" (fnts-scal font-list))
      ;; (format *dvi-font-infout* "~%Pixel Dir:     ~S" (fnts-dpos font-list))
      ;; (format *dvi-font-infout* "~%Font Count:    ~S" (fnts-leng font-list))
      ;;
      ;; 各文字の pixel 情報を取り出す。
      ;;
```

```
    ;; (setf time2 (zl:time))
    ;; (format *dvi-font-infout* " ~A ~A" (float (/ (- time2 time1) 60)))
  font-list))
```

CLM01:>local>system>dvi-previewer>show-previewer.lisp.3

For: Kiyoshi Hayashi
Printed on: comprinter
Number of copies: 1
Data created at: 7/31/90 12:54:15
Queued at: 7/31/90 13:02:48

```
;;; -*- Mode: LISP; Syntax: Common-lisp; Base: 10; Package: DVI -*-

(defmacro mm-dvi (mm) '(round (/ (* ,mm 473628672) 2540)))

(defvar *dvi-prev-infout* *standard-output*)
;(defvar *dvi-paper-size* :a4)
(defvar *dvi-paper-face* :port-rate)
(defvar *dvi-a4-short* 209)
(defvar *dvi-a4-long*  297)
(defvar *dvi-b4-short* 256)
(defvar *dvi-b4-long*  363)

(defun dvi-previewer (file-name &optional (window *standard-output*))
  (let ((page-no 0)
        (paper-width)
        (paper-height)
        (top-margin (mm-dvi *dvi-top-margin*))
        (left-margin (mm-dvi *dvi-left-margin*)))
    ;; (format t "~%DVI-PREVIEWER ~S" file-name)
    (cond
      ((and (eq :a4 *dvi-paper-size*) (eq :port-rate *dvi-paper-face*))
       (setf paper-width  (mm-dvi *dvi-a4-short*))
       (setf paper-height (mm-dvi *dvi-a4-long*)))
      ((and (eq :a4 *dvi-paper-size*) (eq :land-scope *dvi-paper-face*))
       (setf paper-width  (mm-dvi *dvi-a4-long*))
       (setf paper-height (mm-dvi *dvi-a4-short*)))
      ((and (eq :b4 *dvi-paper-size*) (eq :port-rate *dvi-paper-face*))
       (setf paper-width  (mm-dvi *dvi-b4-short*))
       (setf paper-height (mm-dvi *dvi-b4-long*)))
      ((and (eq :b4 *dvi-paper-size*) (eq :land-scope *dvi-paper-face*))
       (setf paper-width  (mm-dvi *dvi-b4-long*))
       (setf paper-height (mm-dvi *dvi-b4-short*))))
    (with-open-file (ifp file-name :direction :input :characters nil :byte-size 8)
      (do ((com-byte (read-unsigned-bytes ifp 1) (read-unsigned-bytes ifp 1))
           (num 0)                              ; Numerator
           (den 0)                              ; Denominator
           (mag 0)                              ; Magnification
           (fonts-tab nil)                      ; Fonts Table ((F# Mag Pxl) ... )
           (cur-f nil)                          ; Current Font Information
           (h 0) (v 0) (w 0) (x 0) (y 0) (z 0)
           (stack nil)                          ; Registers Stack
           (nc) (nh) (nv))
          ((eq nil com-byte))
        ;; (if (> com-byte 127) (format *dvi-prev-infout* " ~S" com-byte))
        (cond
          ;;
          ;; Set Character
          ;;
          ((<= 0 com-byte 127)
           (if (write-font window (dvi-dot h) (dvi-dot v) com-byte cur-f)
               (incf h (pxls-tfmw (aref (fnts-char cur-f) com-byte)))))
          ;;
          ;; Set #
          ;;
          ((<= 128 com-byte 131)
           (setf nc (read-unsigned-bytes ifp (- com-byte 127)))
           (if (write-font window (dvi-dot h) (dvi-dot v) nc cur-f)
               (incf h (pxls-tfmw (aref (fnts-char cur-f) nc)))))
          ;;
          ;; Set Rule
          ;;
          ((= 132 com-byte)
           (let ((nv) (nh) (lef) (top) (rig) (bot))
             (setf nv (read-unsigned-bytes ifp 4))
             (setf nh (read-unsigned-bytes ifp 4))
             ;; (format *dvi-prev-infout* "~%SetRule")
             (setf lef (dvi-dot h))
             (setf top (dvi-dot (- v nv)))
             (setf rig (dvi-dot (+ h nh)))
             (setf bot (dvi-dot v))
             (if (= lef rig)
                 (if (plusp nh)
                     (incf rig)
                     (decf rig)))
             (if (= top bot)
                 (if (plusp nv)
                     (incf bot)
                     (decf bot)))
```

```
    (graphics:draw-rectangle lef top rig bot :stream window)
    (incf h nh)))
;;
;; Put #
;;
((<= 133 com-byte 136)
 (setf nc (read-unsigned-bytes ifp (- com-byte 132)))
 (write-font window (dvi-dot h) (dvi-dot v) nc cur-f))
;;
;; Put Rule
;;
((= 137 com-byte)
 (let ((nv) (nh) (lef) (top) (rig) (bot))
   (setf nv (read-unsigned-bytes ifp 4))
   (setf nh (read-unsigned-bytes ifp 4))
   ;; (format *dvi-prev-infout* "~%PutRule")
   (setf lef (dvi-dot h))
   (setf top (dvi-dot (- v nv)))
   (setf rig (dvi-dot (+ h nh)))
   (setf bot (dvi-dot v))
   (if (= lef rig)
       (if (plusp nh)
           (incf rig)
           (decf rig)))
   (if (= top bot)
       (if (plusp nv)
           (incf bot)
           (decf bot)))
   (graphics:draw-rectangle lef top rig bot :stream window)))
;;
;; No Operation
;;
((= 138 com-byte) nil)
;;
;; Begin of Page
;;
((= 139 com-byte)
 ;; (format *dvi-prev-infout* "~%BOP ~S ~S ~S ~S ~S"
 ;;       page-no paper-width paper-height top-margin left-margin)
 (dotimes (i 10)
   (read-unsigned-bytes ifp 4))
 (read-unsigned-bytes ifp 4)
 (setf h 0)
 (setf v (* page-no paper-height))
 (setf w 0)
 (setf x 0)
 (setf y 0)
 (setf z 0)
 (setf stack nil)
 (setf cur-f nil)
 (graphics:draw-rectangle
   (dvi-dot h)                          ; left
   (dvi-dot v)                          ; top
   (dvi-dot (+ h paper-width))          ; right
   (dvi-dot (+ v paper-height))         ; bottom
   :stream window
   :filled nil)
 (incf h left-margin)
 (incf v top-margin)
 (incf page-no))
;;
;; End of Page
;;
((= 140 com-byte) nil)
;;
;; Push
;;
((= 141 com-byte)
 (push (list h v w x y z) stack))
;;
;; Pop
;;
((= 142 com-byte)
 (setf nc (pop stack))
 (setf h (elt nc 0))
 (setf v (elt nc 1))
 (setf w (elt nc 2))
 (setf x (elt nc 3))
```

```
 (setf y (elt nc 4))
 (setf z (elt nc 5)))
;;
;; Right #
;;
((<= 143 com-byte 146)
 (setf nh (read-signed-bytes ifp (- com-byte 142)))
 (incf h nh))
;;
;; W 0
;;
((= 147 com-byte)
 (incf h w))
;;
;; W #
;;
((<= 148 com-byte 151)
 (setf w (read-signed-bytes ifp (- com-byte 147)))
 (incf h w))
;;
;; X 0
;;
((= 152 com-byte)
 (incf h x))
;;
;; X #
;;
((<= 153 com-byte 156)
 (setf x (read-signed-bytes ifp (- com-byte 152)))
 (incf h x))
;;
;; Down #
;;
((<= 157 com-byte 160)
 (setf nv (read-signed-bytes ifp (- com-byte 156)))
 (incf v nv))
;;
;; Y 0
;;
((= 161 com-byte)
 (incf v y))
;;
;; Y #
;;
((<= 162 com-byte 165)
 (setf y (read-signed-bytes ifp (- com-byte 161)))
 (incf v y))
;;
;; Z 0
;;
((= 166 com-byte)
 (incf v z))
;;
;; Z #
;;
((<= 167 com-byte 170)
 (setf z (read-signed-bytes ifp (- com-byte 166)))
 (incf v z))
;;
;; Set Font Number
;;
((<= 171 com-byte 234)
 (setf nc (- com-byte 171))
 (dolist (font fonts-tab)
   (if (= nc (car font))
       (let ()
         (setf cur-f (caddr font))
         ;; (format *dvi-prev-infout* "~%Font Select ~S ~S ~% ~S"
         ;; (car font) (cadr font) cur-f)
         (return-from nil)))))
;;
;; Set Font #
;;
((<= 235 com-byte 238)
 (setf nc (read-unsigned-bytes ifp (- com-byte 234)))
 (dolist (font fonts-tab)
   (if (= nc (car font))
       (let ()
```

```
          (setf cur-f (caddr font))
          (return-from nil)))))
;;
;; Xxx # (\special)
;;
((<= 239 com-byte 242)
 (setf nc (read-unsigned-bytes ifp (- com-byte 238)))
 (dotimes (i nc)
   (read-unsigned-bytes ifp 1)))
;;
;; Define Font #
;;
((<= 243 com-byte 246)
 (let ((fn) (sf) (ds) (a) (l) (n "") (no))
   (setf fn (read-unsigned-bytes ifp (- com-byte 242)))
   (read-unsigned-bytes ifp 4)          ; Check Sum
   (setf sf (read-unsigned-bytes ifp 4))      ; Scale Factor
   (setf ds (read-unsigned-bytes ifp 4))      ; Design Size
   (setf a (read-unsigned-bytes ifp 1))
   (setf l (read-unsigned-bytes ifp 1))
   (if (plusp a)
       (read-unsigned-bytes ifp a))   ; Directory
   (dotimes (i l)
     (setf n (format nil "~A~A" n (int-char(read-unsigned-bytes ifp 1)))))
   (setf no (round (/ (* sf *dvi-resolution* 5) ds)))
   (setf no (case no
              (1642 1643)
              (3111 3110)
              (otherwise no)))
   (setf n (format nil "~A.~Apxl" n no))
   ;; (format *dvi-prev-infout* "~% Font Name ~S" n)
   (do ((fonts fonts-tab (cdr fonts))
        (font-dat))
       ((null fonts)
        (setf font-dat (make-font n sf))
        (push (list fn sf font-dat) fonts-tab))
     (if (= fn (caar fonts))
         (return-from nil)))))
;;
;; Beginning of the Preamble
;;
((= 247 com-byte)
 (read-unsigned-bytes ifp 1)          ; DVI Version
 (setf num (read-unsigned-bytes ifp 4))       ; Numerator
 (setf den (read-unsigned-bytes ifp 4))       ; Denominator
 (setf mag (read-unsigned-bytes ifp 4))       ; Magnification
 (setf nc (read-unsigned-bytes ifp 1))
 (dotimes (i nc)
   (read-unsigned-bytes ifp 1)))
;;
;; Beginning of the Postamble
;;
((= 248 com-byte)
 (read-unsigned-bytes ifp 4)          ; Pointer to Final BOP
 (read-unsigned-bytes ifp 4)          ; Numerator
 (read-unsigned-bytes ifp 4)          ; Denominator
 (read-unsigned-bytes ifp 4)          ; Magnification
 (read-unsigned-bytes ifp 4)          ; Height of Tallest Page
 (read-unsigned-bytes ifp 4)          ; Width of Widest Page
 (read-unsigned-bytes ifp 2)          ; Maximum Stack Depth
 (read-unsigned-bytes ifp 2))         ; Total Number of Page
;;
;; Ending of the Postamble
;;
((= 249 com-byte)
 (read-unsigned-bytes ifp 4)          ; Pointer to POST
 (do ((nc (read-unsigned-bytes ifp 1) (read-unsigned-bytes ifp 1)))
     ((eq nil nc))))
;;
;; Unknown
;;
((<= 250 com-byte 255) nil)
)))))
```