DIRECTORY

BitBltDefs: FROM "bitbltdefs" USING [BBptr, BBTable, BITBLT],
FontDefs: FROM "fontdefs" USING [BitmapState, FontHandle, FontObject],
InlineDefs: FROM "inlinedefs" USING [BITAND, BITOR, BITSHIFT],
SegmentDefs: FROM "segmentdefs" USING [
    FileSegmentAddress, FileSegmentHandle, SwapIn, SwapOut, Unlock],
SystemDefs: FROM "systemdefs" USING [AllocateHeapNode, FreeHeapNode];

DEFINITIONS FROM FontDefs;

A1Font: PROGRAM IMPORTS SegmentDefs, SystemDefs EXPORTS FontDefs =
BEGIN

FileSegmentHandle: TYPE = SegmentDefs.FileSegmentHandle;

CR: CHARACTER = 15C;
SP: CHARACTER = ' ';

A1FontObject: TYPE = RECORD [procs: FontObject,
   seg: FileSegmentHandle,
   lockCount: CARDINAL,
   height: CARDINAL];

A1FontHandle: TYPE = POINTER TO A1FontObject;

FHptr: TYPE = POINTER TO FontHeader;
Fptr: TYPE = POINTER TO Font;
FCDptr: TYPE = POINTER TO FCD;
FAptr: TYPE = POINTER TO FontArray;
FontArray: TYPE = ARRAY [0..255] OF FCDptr;

Font: TYPE = MACHINE DEPENDENT RECORD [
   header: FontHeader,
   FCDptrs: FontArray, -- array of self-relative pointers to
      -- FCD's. Indexed by char value.
   extFCDptrs: FontArray -- array of self-relative pointers to
      -- FCD's for extentsions. As large as
      -- array as needed.
];

FontHeader: TYPE = MACHINE DEPENDENT RECORD [
   maxHeight: CARDINAL, -- height of tallest char in font (scan lines)
   variableWidth: BOOLEAN, -- IF TRUE, proportionally spaced font
   blank: [0..177B], -- not used
   maxWidth: [0..377B] -- width of widest char in font (raster units).
];

FCD: TYPE = MACHINE DEPENDENT RECORD [
   widthOrExt: [0..77777B], -- width or extention index
   hasNoExtension: BOOLEAN, -- TRUE=> no ext.; prevfield=width
   height: [0..377B], -- # scan lines to skip for char
   displacement: [0..377B] -- displacement back to char bitmap
];

CharWidth: PUBLIC PROCEDURE [font: FontHandle, char: CHARACTER] RETURNS [w: CARDINAL] =
BEGIN
   code: CARDINAL;
   cw: FCDptr;
   fontdesc: FAptr;
   -- check for control characters
   IF char = CR THEN char = SP;
   IF char < SP THEN
      RETURN[CharWidth[font, '+' ] +
            CharWidth[font, LOOPHOLE[LOOPHOLE[char,CARDINAL]+100B,CHARACTER] ] ];
   w = 0;
   fontdesc = @LockFont[font].FCDptrs;
   code = LOOPHOLE[char];
   DO cw = LOOPHOLE[fontdesc[code]+LOOPHOLE[fontdesc,CARDINAL]+code];
      IF cw.hasNoExtension THEN EXIT;
   END CharWidth;
w ← w+16;
code ← cw.widthORext;
ENDLOOP;
w ← w+cw.widthORext;
UnlockFont[font];
RETURN
END;

CharHeight: PUBLIC PROCEDURE [font: FontHandle, char: CHARACTER] RETURNS [CARDINAL] =
BEGIN
RETURN[LOOPHOLE[font,AlFontHandle].height]
END;

PaintChar: PROCEDURE
[font: FontHandle, char: CHARACTER, bmState: POINTER TO BitmapState] =
BEGIN
OPEN BitBitDefs, bmState;
bba: ARRAY [0..SIZE[BBTable]] OF UNSPECIFIED;
btt: BBptr = LOOPHOLE[BASE[bba] + LOOPHOLE[BASE[bba],CARDINAL] MOD 2];
cw: FCDptr;
fontdesc: FAptr = @LockFont[font].FCDptrs;
code: CARDINAL + LOOPHOLE[char];
bbtr ← [
pad: 0,
sourcealt: FALSE,
destalt: FALSE,
sourcectype: block,
function: paint,
unused:,
dbca: origin,
dbmr: wordsPerLine,
dlx: x,
dty:,
dw: 16,
dh:,
sbca:,
sbmnr: 1,
six: 0,
sty: 0,
gray0: gray1:, gray2:, gray3:];
DO
cw ← LOOPHOLE[fontdesc[code]+LOOPHOLE[fontdesc,CARDINAL]+code];
bbt.dty ← y + cw.height;
bbt.dh ← cw.displacement;
bbt.sbca ← cw - (bbt.dh + cw.displacement);
IF cw.hasNoExtension THEN
BEGIN
x ← x + (bbt.dw ← cw.widthORext);
BITBLT[bbt];
EXIT
END
ELSE
BEGIN
BITBLT[bbt];
bbt.dlx ← x ← x + 16;
END;
code ← cw.widthORext;
ENDLOOP;
UnlockFont[font];
RETURN
END;

ClearChar: PROCEDURE
[font: FontHandle, char: CHARACTER, bmState: POINTER TO BitmapState] =
BEGIN
OPEN bmState, InlineDefs;
bit: [0..15];
xword: CARDINAL;
scanLines: CARDINAL = LOOPHOLE[font,AlFontHandle].height;
start, p: POINTER;
cwidth: INTEGER + CharWidth[font,char];
mask: WORD;
ones: WORD = 177777B;
IF x < cwidth THEN BEGIN cwidth ← x; x ← 0 END
ELSE x ← x - cwidth;
xword ← x/16;
x ← x MOD 16;
mw + BITOR[BITSHIFT[ones,16-bit],BITSHIFT[ones,-(bit+cwidth)]];
start + origin + xword + y*wordsPerLine-1;
cwidth + cwidth + bit;
DO
  p ~ start ~ start + 1;
  THROUGH [0..scanLines] DO
    p ~ BITAND[p, mask];
    p ~ p + wordsPerLine;
    ENDLOOP;
    IF (cwidth ~ cwidth - 16) <= 0 THEN EXIT;
    mask ~ BITSHIFT[ones, -cwidth];
    ENDLOOP;
    RETURN
END;

LockFont: PROCEDURE [font: FontHandle] RETURNS [Fptr] ~
BEGIN
  OPEN SegmentDefs, af: LOOPHOLE[font, A1FontHandle];
  IF (af.lockCount ~ af.lockCount + 1) = 1 THEN SwapIn[af.seg];
  RETURN[FileSegmentAddress[af.seg]]
END;

UnlockFont: PROCEDURE [font: FontHandle] ~
BEGIN
  OPEN SegmentDefs, af: LOOPHOLE[font, A1FontHandle];
  IF (af.lockCount ~ af.lockCount - 1) = 0 THEN Unlock[af.seg];
  RETURN
END;

DestroyFont: PROCEDURE [font: FontHandle] ~
BEGIN
  CloseFont[font];
  SystemDefs.FreeHeapNode[font];
  RETURN
END;

CloseFont: PROCEDURE [font: FontHandle] ~
BEGIN
  OPEN af: LOOPHOLE[font, A1FontHandle];
  IF af.seg.lock ~ 0 THEN SegmentDefs.SwapOut[af.seg];
  RETURN
END;

CreateFont: PUBLIC PROCEDURE [fontSegment: FileSegmentHandle] RETURNS [f: FontHandle] ~
BEGIN
  f ~ LOOPHOLE[p];
  p ~ E
    proc: [
      paintChar: PaintChar,
      clearChar: ClearChar,
      charWidth: CharWidth,
      charHeight: CharHeight,
      close: CloseFont,
      destroy: DestroyFont,
      lock: LockFont,
      unlock: UnlockFont],
    seg: fontSegment,
    lockCount: 0,
    height: LockFont[f].header.maxHeight];
  UnlockFont[f];
  RETURN
END.