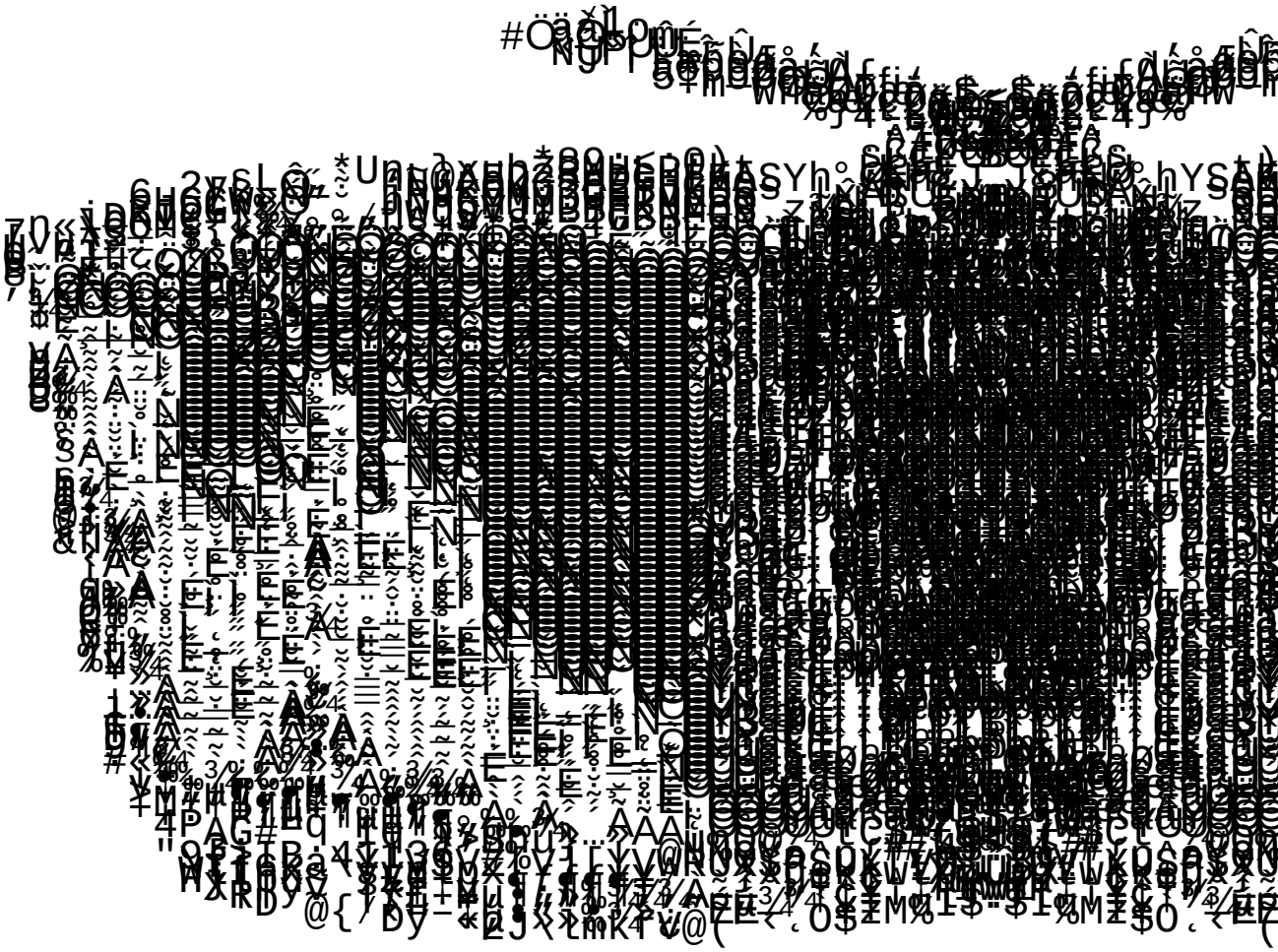

Straw
Program

Alto/Mesa Program: Straw

Joe Maleson

August 5, 1980

This document filed on [IVY]<Maleson>Straw.PRESS

[IVY]<Maleson>Straw.BCD is a Mesa 6 system for manipulating Press files with a SIL-like editor. It works on Altos or D-machines, but performance on Altos is not spectacular. Each Press object (text string, rectangle, cubic section, bitmap, etc.) can be selected, moved, copied, or deleted using the subset of SIL editing commands described below. In addition, special functions are available for dealing with 8-bit per point images.

Input Press files are selected from a menu containing the names of all files ending in ".press" on your local disk. For multi-page input files, a page number menu will be displayed. A menu item is selected by bugging the red (left or top) mouse button over an item, and then moving the mouse outside of the menu. The menu can be moved by holding down the yellow (middle) mouse button, and moving the mouse.

SIL Commands

Two position bars are displayed: a vertical bar shows the MARK position, and a horizontal bar shows the ORIGIN position. Operations are initiated by combinations of mouse buttons and the <CTRL> and left <SHIFT> keys, or by typing certain control characters.

SIL names for the mouse buttons are Mark (left, top, or red), Draw (middle or yellow), and Select (right, bottom, or blue). The mouse button functions are as follows (non-standard implementation is shown in boldface):

Mark:	Move the MARK here
with Shift:	Move the ORIGIN here
with Ctrl:	Move selected objects so the current ORIGIN moves here
with Ctrl/Shift:	**Magnify window turned on and moved here**
Draw:	**Show command summary**
with Shift:	Delete the object pointed to
with Ctrl:	Copy selected objects so the current ORIGIN moves here
with Ctrl/Shift:	Undelete the last group of things deleted (up to 5 levels)
Select:	Select the object pointed to
with Shift:	Select every object entirely contained in the area between the MARK and here
with Ctrl:	Select this object without deselecting previous ones
with Ctrl/Shift:	Deselect the object pointed to

The control characters currently implemented are:

^C: Copy (just like Draw+Ctrl)
^D: Delete all selected items
^I: Input a file (Press file menu appears)
^K: Reinitialize (Clear screen)
^M: Turn magnify window on/off at MARK
^O: Output to file
^Q: Quit
^T: "Ticks": puts up a background grid after requesting the inter-dot spacing in points
^U: Undelete the last group of things deleted (up to 5 levels) (just like Ctrl+Shift+Draw)

Magnify Window

The magnify window allows you to view sections of the Press file at resolutions higher or lower than normal (adjustable from 10 bits/inch to 2540 bits/inch). At the bottom of the magnify window is a magnification control area and a window size control area. Current magnification is indicated by a vertical bar, and is changed by moving the mouse into the magnification control area and clicking the "Mark" button. The magnification control icon is a black triangle widening to the right: the steps of the triangle are located at useful magnifications (72,192,384,723,1270 bits/inch). To the right of the magnification control area is a the window size control area, indicated by a small square. To change the window size, move the mouse into the control area, and hold down the "Mark" button. As the mouse moves, the new window size is indicated by a flashing rectangle. Letting up on the "Mark" button sets the window size to that of the current rectangle. The magnification window is moved by pointing to the location of the center of the new window, and clicking Ctrl+Shift+Mark. Note that the MARK and ORIGIN bars are displayed in unmagnified coordinates, and will generally not correspond to the actual position of MARK and ORIGIN in the magnify window.

Continuous Tone Image Manipulatione

Straw provides some special editing functions for 8-bit per point images. The special functions are available when a single 8-bit per point image is selected, but are disabled when the magnify window is displayed. Functions are invoked by placing the cursor in one of three concentric "ring" areas, starting at the outside edge of the image and working toward the center. In an active ring area, the cursor will be displayed on a white background, and will take on a shape indicating its function; pressing the Select mouse button will initiate the function.

Rescaling: The rescaling area is outermost ring around of the image. When the cursor is above an *outside corner* of the selected image, it will be displayed as a diagonal double-arrow; holding down the Select mouse button will allow you to rescale the image by moving the designated corner. This rescaling will retain the original aspect ratios of the image. When the cursor is above an *outside edge* of the selected image, it will be displayed as a horizontal or vertical

double-arrow; holding down the Select button will allow you to stretch or shrink the image by moving the designated edge. This action will destroy the original aspect ratios of the image, and produce anamorphic distortion. When the button is released, the image will be re-halftoned at the indicated size.

Cropping: The cropping (or windowing) area is the second ring, just inside the edge of the image, and is indicated by a cursor containing a square, with a double arrow on a side or corner of the square. Holding down the Select button in a corner cropping area will allow you to move two edges of the cropping window; holding the select button over a horizontal or vertical side, you will be able to move a single edge of the cropping window. When the button is released, the indicated area of the image will be re-halftoned at the new size. After an image has been cropped, the cropping window can be expanded (up to the original size of the image). Note that any output Press files will *not* contain the image areas which have been cropped out.

Rotation: The rotation/mirroring area is the third ring, closest to the center of the image. The four functions are "rotate 90 ^{o}", "rotate 180 ^{o}", "rotate 270 ^{o}", and "mirror", in that order beginning at the top right corner and proceding counterclockwise around the ring. Note that on an Alto, the rotated/mirrored halftone operations are not microcoded, and will be exceedingly slow.

Contrast Enhancement: The contrast enhancement control panel is invoked by pressing the Draw button while the cursor is over any area of the selected image. This control panel consists of two gray wedges, and black and white "flags" indicating the current values used for black and white. Holding the Mark button while the cursor is on one of the flags allows you to move that flag position. The top gray wedge is a fixed scale from black to white. The bottom gray wedge shows the output values for each pixel in the range 0-256. Positioning the flags on the bottom wedge maps any value less than the black position to black, and any value greater than the white position to white. Positioning a flag on the top wedge converts the minimum (black) or maximum (white) value to the indicated intermediate gray level. When the cursor is moved outside of the control panel area, the image will be rehalftoned using the new black and white levels.

CAUTION

Do not try to write over one of the Press files you have used as input for the current page. Straw does not currently check for this, and it will cause strange and hard to predict behavior.

Easy Improvements

Color control (like SIL's ^J)
Expand 8-bit editing functions to work on any "Dots" object
The image manipulation interface is quite ad-hoc, and probably non-optimal. It would be easy to try some other interfaces.

```
Straw: CONFIGURATION
    IMPORTS DirectoryDefs, DiskDefs, IODefs, MiscDefs, ProcessDefs, SegmentDefs, SystemDefs, StreamDefs, StringDefs, TimeDefs
    CONTROL Straw0 =
BEGIN
Graphics;
PressIO;
Halftone;

Magic;
DisplayList;
XMAlloc;
StrawSil;
StrawMagnify;
StrawImage;
Straw1;
Straw0;
END.
```

```
DIRECTORY
    GraphicsDefs: FROM "GraphicsDefs",
    AltoFileDefs: FROM "AltoFileDefs",
    PressDefs: FROM "PressDefs";

StrawDefs: DEFINITIONS =
BEGIN

EntityBox : TYPE = RECORD
[   Box: GraphicsDefs.Box,
    Object: POINTER TO PressDefs.PressObject
];

DotObject: TYPE = RECORD
[   Box: GraphicsDefs.Box,
    Object: POINTER TO PressDefs.PressObject,
    DotsData: POINTER TO PressDefs.PressDotsData
];

RectangleParams  : TYPE = RECORD
[   micaWidth,micaHeight: CARDINAL
];

FontParams  : TYPE = RECORD
[   spaceX,font: CARDINAL
];

FontListEntry  : TYPE = RECORD
[   link: POINTER TO FontListEntry,
    n: CARDINAL,
    strike: POINTER TO GraphicsDefs.StrikeFont,
    mode: GraphicsDefs.textMode,
    width: POINTER TO ARRAY CHARACTER [0C..255C] OF CARDINAL,
    index: CARDINAL,
    PressFontEntry: PressDefs.FontEntry
];

WaitCursor  : PUBLIC PROCEDURE;

NormalCursor  : PUBLIC PROCEDURE;

Refresh : PROCEDURE;

GetNum : PUBLIC PROCEDURE [prompt: STRING] RETURNS [CARDINAL];

GetNumWithDefault  : PUBLIC PROCEDURE [prompt: STRING,default: CARDINAL] RETURNS [CARDINAL];

GetString : PUBLIC PROCEDURE [s,prompt: STRING];

CreateDotsBox  : PUBLIC PROCEDURE [do: POINTER TO DotObject];

CreateObjectBox  : PUBLIC PROCEDURE [eBox: POINTER TO EntityBox];

DisplayCoord  : PROCEDURE [x,y: INTEGER] RETURNS [dispX,dispY: INTEGER];

PageCoord : PROCEDURE [dispX,dispY: INTEGER] RETURNS [x,y: INTEGER];

ReleaseEntityBox  : PROCEDURE [eBox: POINTER TO EntityBox,erase: BOOLEAN];

SetGrid : PUBLIC PROCEDURE;

RestoreGrid  : PROCEDURE [currentBox: POINTER TO GraphicsDefs.Box];

DisplayPressPage  : PROCEDURE [p: POINTER TO PressDefs.PressPage];

SelectFile  : PUBLIC PROCEDURE;

AddFile : PUBLIC PROCEDURE;

NewPage  : PUBLIC PROCEDURE;

WriteFile  : PUBLIC PROCEDURE;
```

EditDisplayList : PROCEDURE ;

EditGrayImage : PROCEDURE [do: POINTER TO DotObject];

GetFont : PROCEDURE [n: CARDINAL] RETURNS [strike: POINTER TO GraphicsDefs.StrikeFont,mode: GraphicsDefs.textMode,width: POINTER TO ARRAY CHARACTER [0C..255C] OF CARDINAL,index: CARDINAL];

GetPressFontEntry : PROCEDURE [index: CARDINAL] RETURNS [POINTER TO PressDefs.FontEntry];

GetFontListHandle : PROCEDURE RETURNS[POINTER TO FontListEntry];

ReleaseFontStorage : PROCEDURE;

PressPageFromDisplayList : PROCEDURE [file: STRING];

--from ScrapMagnify

StartMagnifier : PROCEDURE [Index,Gray: CARDINAL];
ShowMagnifier : PROCEDURE;
StopMagnifier : PROCEDURE;
SetMagnification : PROCEDURE [bitsPerInch: CARDINAL];
SetMagnifierPosition : PROCEDURE [x,y: CARDINAL];
SetMagnifierSize : PROCEDURE [width,height: CARDINAL];
ModifyMagnifier : PROCEDURE [x,y: CARDINAL] RETURNS [BOOLEAN];
MagToScreenCoords : PROCEDURE [x,y: CARDINAL] RETURNS [rx,ry: CARDINAL];
MagToMicaCoords : PROCEDURE [x,y: CARDINAL] RETURNS [rx,ry: CARDINAL];
ScreenToMagCoords : PROCEDURE [x,y: CARDINAL] RETURNS [rx,ry: CARDINAL];
END.

```
DIRECTORY
    AltoFileDefs: FROM "AltoFileDefs",
    AltoMicrocodeDefs: FROM "AltoMicrocodeDefs",
    DiscoDefs: FROM "DiscoDefs" USING[Lookup,LookupList,TransferBytes],
    DisplayListDefs: FROM "DisplayListDefs",
    GraphicsDefs: FROM "GraphicsDefs",
    ImageDefs: FROM "ImageDefs",
    InlineDefs: FROM "InlineDefs" USING [COPY],
    IODefs: FROM "IODefs" USING [WriteLine,WriteString,ReadID,Rubout],
    LongDefs: FROM "LongDefs" USING [InitMachineType],
    MagicDefs: FROM "MagicDefs",
    MiscDefs: FROM "MiscDefs",
    Mopcodes: FROM "Mopcodes",
    PressDefs: FROM "PressDefs" USING
        [PressPage,ReleasePressPage,PressFileDescriptor,
        InitPressFileDescriptor,PutPressPage,ClosePressFile,PressObject,
        PressFilePointers,PressFont,ELShowDots,ELShowDotsOpaque,
        OpenPressFileFromFP,ReadPressPage,ReleasePressFilePointers],
    StrawDefs: FROM "StrawDefs",
    SegmentDefs: FROM "SegmentDefs",
    StreamDefs: FROM "StreamDefs",
    StringDefs: FROM "StringDefs",
    SystemDefs: FROM "SystemDefs",
    XMAllocDefs: FROM "XMAllocDefs";

Straw0 : PROGRAM IMPORTS AltoMicrocodeDefs, DiscoDefs, DisplayListDefs, ImageDefs, InlineDefs, IODefs, GraphicsDefs,
LongDefs, MagicDefs, MiscDefs, PressDefs, StrawDefs, SegmentDefs, StringDefs, SystemDefs, XMAllocDefs
    EXPORTS StrawDefs =
BEGIN
OPEN DiscoDefs,InlineDefs, IODefs, StrawDefs;

Buttons: TYPE = MACHINE DEPENDENT RECORD
    [ KeyPadAndGarbage: [0..10000B),
      Red: [0..1],
      Blue: [0..1],
      Yellow: [0..1]
    ];
triState: TYPE = {empty,singleFile,multiFile};
MouseButtons : POINTER TO Buttons = LOOPHOLE[177030B];
CursorX : POINTER TO CARDINAL = LOOPHOLE[426B];
CursorY : POINTER TO CARDINAL = LOOPHOLE[427B];
CursorBits : POINTER TO ARRAY [0..15] OF CARDINAL = LOOPHOLE[431B];
Cursor : GraphicsDefs.Bitmap _ [bank: 0,nWords: 1,nBits:16,nLines:16,
bits:LOOPHOLE[CursorBits],nBitsPerPixel:1,portraitMode:TRUE,scaleFactor:10];
ArrowCursor: GraphicsDefs.Bitmap _ [bank: 0,nWords: 1,nBits: 16,nLines: 16,
bits:,nBitsPerPixel:1,portraitMode:TRUE,scaleFactor:10];
HourGlassCursor: GraphicsDefs.Bitmap _ [bank: 0,nWords: 1,nBits: 16,nLines:
16,bits:,nBitsPerPixel:1,portraitMode:TRUE,scaleFactor:10];

Screen : POINTER TO GraphicsDefs.Bitmap;

bitsPerInch : CARDINAL _ 72;
micasPerInch : CARDINAL = 2540;
dispXOff : CARDINAL _ 0;
dispYOff : CARDINAL _ 0;
GridSize: CARDINAL _ 0;
DCBHead: POINTER TO CARDINAL = LOOPHOLE[420B];
nextDCB: POINTER TO ARRAY[0..3] OF UNSPECIFIED _ LOOPHOLE[DCBHead^];
nScanLines: CARDINAL _ 808;
DCBtop: POINTER TO ARRAY[0..3] OF CARDINAL _ nextDCB;

pfp: PressDefs.PressFilePointers;
needRelease: BOOLEAN _ FALSE;
pressNames: POINTER TO ARRAY [0..0) OF STRING;
nFiles: CARDINAL _ 0;
pressFiles: POINTER TO DiscoDefs.LookupList _ DiscoDefs.Lookup["*.press"];
font: POINTER TO GraphicsDefs.StrikeFont _ GraphicsDefs.GetStrikeHandle["SysFont"];
nPages: CARDINAL;
displayPage: CARDINAL;
pressPage: PressDefs.PressPage;
ScreenState: triState _ empty;
```

```
Main : PROCEDURE =
BEGIN
p: POINTER TO LookupList;
--mem defs
TotalPagesForXMem: CARDINAL = 400;
nXMemBlocks: CARDINAL = 4;
PagesPerXMemBlock: CARDINAL = TotalPagesForXMem/nXMemBlocks;

ArrowBits: ARRAY [0..15] OF CARDINAL;
HourGlassBits: ARRAY [0..15] OF CARDINAL _
 [      177777B, 100001B, 040002B, 034034B,
        017170B, 007660B, 003740B, 001700B,
        001100B, 002440B, 004220B, 010610B,
        021704B, 047762B, 177777B, 177777B
 ];

COPY[from: Cursor.bits,nwords: 16,to: @ArrowBits];
ArrowCursor.bits _ LOOPHOLE[@ArrowBits];
HourGlassCursor.bits _ LOOPHOLE[@HourGlassBits];

UNTIL nextDCB[0] = 0 DO
nScanLines _ nScanLines - nextDCB[3]*2;
nextDCB _ LOOPHOLE[nextDCB[0]];
ENDLOOP;
nScanLines _ nScanLines - nextDCB[3]*2;

pressPage.leftX _ pressPage.bottomY _ 77777B;
pressPage.rightX _ pressPage.topY _ 0;
pressPage.ObjectList _ NIL;
pressPage.FontDir _ ALL[NIL];

p_pressFiles;
UNTIL p = NIL DO nFiles _ nFiles + 1;p _ p.link;ENDLOOP;
pressNames _ SystemDefs.AllocateHeapNode[nFiles];
p_pressFiles;nFiles _ 0;
UNTIL p = NIL DO
        pressNames[nFiles] _ p.name;
        nFiles _ nFiles + 1;
        p _ p.link;
        ENDLOOP;

--now, initialize mem
LongDefs.InitMachineType[]; --needed for Alto compatible XMAlloc stuff
AltoMicrocodeDefs.LoadRam[]; --ditto
THROUGH [1..nXMemBlocks] DO
        XMAllocDefs.AddToXMZone[SegmentDefs.LongDataSegmentAddress[
              SegmentDefs.NewDataSegment[SegmentDefs.DefaultXMBase,PagesPerXMemBlock]],
                    PagesPerXMemBlock*256];
ENDLOOP;
GraphicsDefs.SetXMAlloc[XMAllocDefs.Allocate,XMAllocDefs.Free];

Screen _ GraphicsDefs.TurnOnGraphics[];
nextDCB _ LOOPHOLE[DCBHead^];

SelectFile[];
AddFile[];

--menu-less edit
IODefs.WriteLine["Straw of July 14, 1980"];
StrawDefs.EditDisplayList[];
ImageDefs.StopMesa[];

DO --edit loop
BEGIN
commandIndex: CARDINAL;
newFile: CARDINAL = 0;
addFile: CARDINAL = newFile+1;
writeFile: CARDINAL = addFile+1;
edit: CARDINAL = writeFile+1;
quit: CARDINAL = edit+1;
nextPage: CARDINAL = quit+1;
commandNames: ARRAY [0..nextPage] OF STRING _
```

```
        ["New File","Add File","Write File","Edit","Quit","Next Page"];
    nCommands: CARDINAL _ nextPage;
    IF (nPages > displayPage) AND (ScreenState = singleFile) THEN nCommands _ nCommands+1;

    DO --get user command
        commandIndex _ MagicDefs.Menu[nCommands,LOOPHOLE[@commandNames],font];
        IF commandIndex IN [0..nCommands) THEN EXIT;
    ENDLOOP;

    SELECT commandIndex FROM
        newFile,nextPage =>
            BEGIN
            NewPage[];
            IF commandIndex = newFile THEN
                BEGIN
                Refresh[];
                SelectFile[];
                END
            ELSE displayPage _ displayPage+1;
            AddFile[];
            END;
        addFile =>
            BEGIN
            SelectFile[];
            AddFile[];
            END;
        edit => StrawDefs.EditDisplayList[];
        writeFile => WriteFile[];
        quit => ImageDefs.StopMesa[];
        ENDCASE;
    END;
    ENDLOOP; --edit loop

    END;

    SelectFile  : PUBLIC PROCEDURE =
    BEGIN
    pressIndex: CARDINAL _ 177777B;
    p: POINTER TO LookupList;

    IF needRelease THEN
        BEGIN
        PressDefs.ReleasePressFilePointers[@pfp];
        END;
    needRelease _ TRUE;

    DO
        pressIndex _ MagicDefs.Menu[nFiles,LOOPHOLE[pressNames],font];
        IF pressIndex IN [0..nFiles) THEN EXIT;
    ENDLOOP;

    WaitCursor[];
    p _ pressFiles;
    THROUGH [1..pressIndex] DO p _ p.link;ENDLOOP;

    nPages _ PressDefs.OpenPressFileFromFP[@p.fp,@pfp];
    NormalCursor[];

    --get page number from user, and display it

    IF nPages > 99 THEN MiscDefs.CallDebugger["too many pages"];
    IF nPages = 1 THEN displayPage _ 1
    ELSE
        BEGIN
        i: CARDINAL;
        numbers: ARRAY [0..100) OF STRING;
        FOR i IN [0..nPages) DO
            numbers[i] _ SystemDefs.AllocateHeapString[2];
            StringDefs.AppendDecimal[numbers[i],i+1];
        ENDLOOP;
        DO
            displayPage_MagicDefs.Menu[nPages,LOOPHOLE[@numbers],font]+1;
```

```
            IF displayPage IN [1..nPages] THEN EXIT;
        ENDLOOP;
        FOR i IN [0..nPages) DO
            SystemDefs.FreeHeapString[numbers[i]];
        ENDLOOP;
        END; --select displayPage
END;

AddFile : PUBLIC PROCEDURE =
BEGIN
obj,savedObj: POINTER TO PressDefs.PressObject;

WaitCursor[];
--add new page to pressPage
savedObj _ pressPage.ObjectList;
pressPage.ObjectList _ NIL;

PressDefs.ReadPressPage[@pfp,displayPage,@pressPage];
DisplayPressPage[@pressPage];

obj _ pressPage.ObjectList;
UNTIL obj.link = NIL DO obj _ obj.link; ENDLOOP;
obj.link _ savedObj;

IF ScreenState = empty THEN ScreenState _ singleFile
ELSE ScreenState _ multiFile;

NormalCursor[];
END;

NewPage : PUBLIC PROCEDURE =
BEGIN
i: CARDINAL;
displayList: POINTER TO ARRAY [0..0) OF POINTER TO StrawDefs.EntityBox _
            LOOPHOLE[DisplayListDefs.GetDisplayListHandle[]];

ReLinkPressPage[@pressPage];
--free storage allocated by ReadDirs
FOR i DECREASING IN [0..DisplayListDefs.GetDisplayCount[]) DO
        ReleaseEntityBox[displayList[i],FALSE];
ENDLOOP;
DisplayListDefs.ResetDisplayCount[];

PressDefs.ReleasePressPage[@pressPage];
pressPage.leftX _ pressPage.bottomY _ 77777B;
pressPage.rightX _ pressPage.topY _ 0;
pressPage.ObjectList _ NIL;
pressPage.FontDir _ ALL[NIL];

--and grab them heap pages
WHILE SystemDefs.PruneHeap[] DO ENDLOOP;

ScreenState _ empty;
END;

WriteFile : PUBLIC PROCEDURE =
BEGIN
pfp: PressDefs.PressFileDescriptor;
pressStr: STRING _ [40];

GetString[pressStr,"Press file name: "];
IF pressStr.length # 0 THEN
        BEGIN
        WaitCursor[];
        ReLinkPressPage[@pressPage];
        PressDefs.InitPressFileDescriptor[@pfp,pressStr];
        PressDefs.PutPressPage[@pfp,@pressPage];
        PressDefs.ClosePressFile[@pfp];
        DiscoDefs.TransferBytes[dp:NIL,nBytes:0,data:NIL,free:TRUE];    --invalidate cache
        NormalCursor[];
        END;
END;
```

```
WaitCursor  : PUBLIC PROCEDURE =
BEGIN
GraphicsDefs.ReplaceBitmap[@HourGlassCursor,0,0,@Cursor];
END;

NormalCursor  : PUBLIC PROCEDURE =
BEGIN
GraphicsDefs.ReplaceBitmap[@ArrowCursor,0,0,@Cursor];
END;

Refresh : PUBLIC PROCEDURE =
BEGIN
i,x,y: CARDINAL;
b: POINTER TO GraphicsDefs.Box;
displayList: POINTER TO ARRAY [0..0) OF POINTER TO GraphicsDefs.Box _
        DisplayListDefs.GetDisplayListHandle[];
GraphicsDefs.EraseArea[0,0,608,808];
IF GridSize # 0 THEN
    BEGIN
    dotLine: GraphicsDefs.Bitmap _ Screen^;
    dotLine.nLines _ 1;
    FOR x _ 0,x+GridSize UNTIL x > dotLine.nBits
        DO GraphicsDefs.PutPoint[x,0];ENDLOOP;
    FOR y _ GridSize,y+GridSize UNTIL y > Screen.nLines
        DO GraphicsDefs.PutBitmap[@dotLine,0,y,Screen];
    ENDLOOP;
    END;
FOR i IN [0..DisplayListDefs.GetDisplayCount[]) DO
    b _ displayList[i];
    GraphicsDefs.DisplayBox[b,b.displayX,b.displayY];
    GraphicsDefs.ReleaseBox[b];
ENDLOOP;
END;

GetFont : PUBLIC PROCEDURE [n: CARDINAL] RETURNS [strike: POINTER TO GraphicsDefs.StrikeFont,mode:
GraphicsDefs.textMode,width: POINTER TO ARRAY CHARACTER [0C..255C] OF CARDINAL,index: CARDINAL] =
BEGIN
font: POINTER TO PressDefs.PressFont _ pressPage.FontDir[n];
fontName: STRING _ [40];
StringDefs.AppendString[fontName,font.Family];
StringDefs.AppendDecimal[fontName,font.PointSize];
strike _ GraphicsDefs.GetStrikeHandle[fontName];
IF strike = NIL THEN
    strike _ GraphicsDefs.GetStrikeHandle["SysFont"];
RETURN[strike,font.Face,font.Widths,n];
END;

GetNum : PUBLIC PROCEDURE [prompt: STRING] RETURNS [CARDINAL] =
BEGIN ENABLE StringDefs.InvalidNumber => RETRY;
s: STRING _ [40];
GetString[s,prompt];
IF s.length = 0 THEN RETURN[0];
RETURN[StringDefs.StringToDecimal[s]];
END;

GetNumWithDefault   : PUBLIC PROCEDURE [prompt: STRING,default: CARDINAL] RETURNS [CARDINAL]=
BEGIN ENABLE StringDefs.InvalidNumber => RETRY;
s: STRING _ [40];
defaultString: STRING _ [8];
StringDefs.AppendDecimal[defaultString,default];
GetStringWithDefault[s,prompt,defaultString];
RETURN[StringDefs.StringToDecimal[s]];
END;

GetStringWithDefault   : PUBLIC PROCEDURE [s,prompt,default: STRING] =
BEGIN OPEN StringDefs;
newPrompt: STRING _ [80];
AppendString[to: newPrompt,from: prompt];
AppendString[to: newPrompt,from: "["];
AppendString[to: newPrompt,from: default];
AppendString[to: newPrompt,from: "]"];
```

```
GetString[s,newPrompt];
IF s.length = 0 THEN AppendString[to: s,from: default];
END;

GetString : PUBLIC PROCEDURE [s,prompt: STRING] =
BEGIN
heightDCB: TYPE = RECORD
 [      longBit: BOOLEAN,
        height: [0..100000B)
 ];
h: POINTER TO heightDCB _ @nextDCB[3];
normalLen: heightDCB _ nextDCB[3];
IF nextDCB # NIL THEN
        BEGIN
        h.height _ MIN[h.height,nScanLines/2];
        nextDCB[0] _ DCBtop;
        END;
        BEGIN ENABLE Rubout => RETRY;
        WriteLine[""];
        WriteString[prompt];
        ReadID[s];
        END; --of ENABLE
IF nextDCB # NIL THEN
        BEGIN
        h^ _ normalLen;
        nextDCB[0] _ 0;
        END;
END;

ReLinkPressPage : PROCEDURE[pressPage: POINTER TO PressDefs.PressPage] =
BEGIN
displayList: POINTER TO ARRAY [0..0) OF POINTER TO StrawDefs.EntityBox _
                        LOOPHOLE[DisplayListDefs.GetDisplayListHandle[]];
nItems: CARDINAL _ DisplayListDefs.GetDisplayCount[];
d: POINTER TO StrawDefs.EntityBox;
i: CARDINAL;

--first, discard invisible items
FOR i DECREASING IN [0..nItems) DO
        d _ displayList[i];
        IF (INTEGER[d.Box.displayX] < 0) OR
                (INTEGER[d.Box.displayY] < 0) OR
                (d.Box.displayX > Screen.nBits) OR
                (d.Box.displayY > Screen.nLines) THEN
                        ReleaseEntityBox[d,TRUE];
ENDLOOP;

nItems _ DisplayListDefs.GetDisplayCount[];
IF nItems = 0 THEN pressPage.ObjectList _ NIL
ELSE
        BEGIN
        pressPage.ObjectList _ displayList[0].Object;
        FOR i IN [0..nItems-1) DO
                displayList[i].Object.link _ displayList[i+1].Object;
        ENDLOOP;
        displayList[nItems-1].Object.link _ NIL;
        END;
END;

ReleaseEntityBox : PUBLIC PROCEDURE [eBox: POINTER TO StrawDefs.EntityBox,erase: BOOLEAN] =
BEGIN
currentBox: POINTER TO GraphicsDefs.Box _ LOOPHOLE[eBox];
do: POINTER TO StrawDefs.DotObject _ LOOPHOLE[eBox];

GraphicsDefs.DestroyBox[currentBox,FALSE];
IF eBox.Object.command IN
        [PressDefs.ELShowDots..PressDefs.ELShowDotsOpaque] THEN
                BEGIN
                IF do.DotsData.fileName # NIL THEN
                        SystemDefs.FreeHeapString[do.DotsData.fileName];
                END;
--erase=FALSE implies that Object will be released by ReleasePressPage
```

```
IF erase THEN
	BEGIN
	SystemDefs.FreeHeapNode[eBox.Object];
	DisplayListDefs.DeleteDisplayItem[currentBox];
	RestoreGrid[currentBox];
	END
ELSE
	GraphicsDefs.ReleaseBox[currentBox];
SystemDefs.FreeHeapNode[currentBox];
END;

SetGrid : PUBLIC PROCEDURE =
BEGIN
GridSize _ GetNum["Dot spacing (72 = 1 inch): "];
Refresh[];
END;

RestoreGrid  : PUBLIC PROCEDURE [currentBox: POINTER TO GraphicsDefs.Box] =
BEGIN IF GridSize = 0 THEN RETURN;
	BEGIN
	x1: CARDINAL _ currentBox.displayX - (currentBox.displayX MOD GridSize);
	x2: CARDINAL _ currentBox.displayX + currentBox.boxBitmap.nBits;
	y1: CARDINAL _ currentBox.displayY;
	y2: CARDINAL _ y1 + currentBox.boxBitmap.nLines;
	x,y: CARDINAL;
	IF INTEGER[x2] < 0 OR INTEGER[y2] < 0 THEN RETURN;
	IF INTEGER[x1] < 0 THEN x1 _ 0;
	IF INTEGER[y1] < 0 THEN y1 _ 0;
	FOR y _ (y1-(y1 MOD GridSize)),y+GridSize UNTIL y > y2
	DO FOR x _ x1,x+GridSize UNTIL x > x2
		DO GraphicsDefs.PutPoint[x,y];ENDLOOP;
	ENDLOOP;
	END;
END;

MakeLongPointer  : PROCEDURE [ptr: POINTER,bank: UNSPECIFIED] RETURNS [LONG POINTER] = MACHINE CODE BEGIN END;

Main[];
END.
```

```
DIRECTORY
    AltoFileDefs: FROM "AltoFileDefs",
    DiscoDefs: FROM "DiscoDefs" USING[DiskPosition,TransferBytes],
    DisplayListDefs: FROM "DisplayListDefs",
    GraphicsDefs: FROM "GraphicsDefs",
    HalftoneDefs: FROM "HalftoneDefs",
    InlineDefs: FROM "InlineDefs",
    MiscDefs: FROM "MiscDefs",
    Mopcodes: FROM "Mopcodes",
    PressDefs: FROM "PressDefs",
    StrawDefs: FROM "StrawDefs",
    SegmentDefs: FROM "SegmentDefs",
    SystemDefs: FROM "SystemDefs";

Straw1 : PROGRAM IMPORTS DiscoDefs, DisplayListDefs, GraphicsDefs, HalftoneDefs, InlineDefs, MiscDefs, PressDefs, SystemDefs,
StrawDefs EXPORTS StrawDefs =
BEGIN
OPEN DiscoDefs;

maxDataBytes: CARDINAL _ 0;
maxObjCommandSize: CARDINAL = 256; --make larger for complex spline display
KLUDGESegs: CARDINAL _ 0; --modify spline display (show points KSegs..3)
errorVec: ARRAY [0..608) OF INTEGER _ ALL[0];

DoHalftone : PROCEDURE [pdd: POINTER TO PressDefs.PressDotsData,b: POINTER TO GraphicsDefs.Bitmap] =
BEGIN
sampleVec: POINTER _ SystemDefs.AllocateSegment[(pdd.nPixels+1)/2];
sMode: CARDINAL _ pdd.mode MOD 4;
normal: BOOLEAN = sMode IN [2..3]; --otherwise, sideways
dp: DiscoDefs.DiskPosition _ pdd.diskPosition;

IF normal THEN
    HalftoneDefs.InitHalftone[0,0,pdd.displayPixels,b.nBits,pdd.min,pdd.max,b,
                                                                pdd.displayLines,b.nLines,pdd.mode]
ELSE
    HalftoneDefs.InitHalftone[0,0,pdd.displayPixels,b.nLines,pdd.min,pdd.max,b,
                                                                pdd.displayLines,b.nBits,pdd.mode];

DiscoDefs.TransferBytes[dp:@dp,data:sampleVec,nBytes:pdd.passPixels];
THROUGH [0..pdd.passLines) DO
    DiscoDefs.TransferBytes[dp:@dp,data:sampleVec,nBytes:pdd.nPixels];
ENDLOOP;

THROUGH [0..pdd.displayLines) DO
    DiscoDefs.TransferBytes[dp:@dp,data:sampleVec,nBytes:pdd.nPixels];
    [] _ HalftoneDefs.PrintHalftoneLine[sampleVec];
ENDLOOP;

SystemDefs.FreeSegment[sampleVec];
END;

Screen: POINTER TO GraphicsDefs.Bitmap _ GraphicsDefs.GetDefaultBitmapHandle[];

bitsPerInch : CARDINAL _ 72;
micasPerInch : CARDINAL = 2540;
dispXOff : CARDINAL _ 0;
dispYOff : CARDINAL _ 0;

--some spline kludges
PressReal : TYPE = MACHINE DEPENDENT RECORD
 [  sign:      BOOLEAN,
    exponent: [0..256),
    m1: [0..177B],
    m2: INTEGER
 ];

MakeLongInteger : PROCEDURE [lowBits,highBits: INTEGER] RETURNS [LONG INTEGER] = MACHINE CODE BEGIN END;
SplitLongInteger : PROCEDURE [LONG INTEGER] RETURNS [lowBits,highBits: INTEGER] = MACHINE CODE BEGIN END;
DoubleShift :PROCEDURE[a: ARRAY [0..2) OF INTEGER,s: INTEGER] RETURNS[INTEGER] =
BEGIN OPEN InlineDefs;
a[0] _ BITSHIFT[a[0],s];
IF s > 0 THEN a[0] _ a[0] + BITSHIFT[a[1],s-16];
```

```
RETURN[a[0]];
END;

RealConvert : PUBLIC PROCEDURE[fake:ARRAY [0..2) OF INTEGER] RETURNS[i: INTEGER]=
BEGIN OPEN InlineDefs;
p: POINTER TO PressReal _ LOOPHOLE[@fake];
neg: BOOLEAN _ p.sign;
exp: INTEGER;

IF fake[0] = 0 THEN RETURN[0];

IF neg THEN [fake[1],fake[0]] _ SplitLongInteger[-MakeLongInteger[fake[1],fake[0]]];
exp _ p.exponent - 200B;
fake[0] _ BITAND[fake[0],177B];
SELECT exp FROM
      <=0=> RETURN[0];        --between 1/2 and 1;
      IN [1..15] => --i _ DoubleShift[fake,exp-7];
            BEGIN
            s: INTEGER _ exp-7;
            i _ BITSHIFT[fake[0],s];
            IF s > 0 THEN i _ i + BITSHIFT[fake[1],s-16];
            END;
      ENDCASE => MiscDefs.CallDebugger["float too big"];

RETURN[IF neg THEN -i ELSE i];
END;

--to transform A,B,C,D coords to Bezier control points:
--
-- x0 y0                        0    0    0    3          Ax   Ay
-- x1 y1                        0    0    1    3          Bx   By
-- x2 y2 _ (1/3)      *    0    1    2    3    *    Cx   Cy
-- x3 y3                        3    3    3    3          Dx   Dy
--
--     x0 _ Dx
-- x1 _ (Cx/3) + Dx
-- x2 _ (Bx/3) + (2*Cx/3) + Dx
-- x3 _ Ax + Bx + Cx + Dx
--
--(same for y)
--
--since the D coords are current position, we just return just 6 offsets
ControlPoints : PUBLIC PROCEDURE [coeff: POINTER TO ARRAY [0..6) OF INTEGER, dx,dy: POINTER TO ARRAY [0..3) OF
INTEGER] =
BEGIN
dx[0] _ coeff[0]/3;
dx[1] _ (coeff[2] + 2*coeff[0])/3;
dx[2] _ coeff[4] + coeff[2] + coeff[0];
dy[0] _ coeff[1]/3;
dy[1] _ (coeff[3] + 2*coeff[1])/3;
dy[2] _ coeff[5] + coeff[3] + coeff[1];
END;

--calculate gray value from hue,sat,brightness
--
--255      black                              fullSatVal
--
--sat
--
--     0          black                              white
--                0          brightness          255
SetGray : PROCEDURE [currentHue,currentSat,currentBr: CARDINAL] RETURNS [currentGray: CARDINAL] =
BEGIN
RED: CARDINAL = 0;
YELLOW: CARDINAL = 40;
GREEN: CARDINAL = 80;
CYAN: CARDINAL = 120;
BLUE: CARDINAL = 160;
MAGENTA: CARDINAL = 200;
RED2: CARDINAL = 240;
rSAT,cSAT: CARDINAL = 64;
ySAT: CARDINAL = 160;
```

```
bSAT: CARDINAL = 32;
fullSatVal,scaledSatVal,gray: CARDINAL;
fullSatVal _ SELECT currentHue FROM
      IN [RED..YELLOW) => rSAT + (currentHue*(ySAT-rSAT))/(YELLOW-RED),
      IN [YELLOW..CYAN) =>
      ySAT - ((currentHue-YELLOW)*(ySAT-cSAT))/(CYAN-YELLOW),
      IN [CYAN..BLUE) => cSAT - ((currentHue-CYAN)*(cSAT-bSAT))/(BLUE-CYAN),
      IN [BLUE..RED2) =>  bSAT + ((currentHue-BLUE)*(rSAT-bSAT))/(RED2-BLUE),
      ENDCASE => 255;
--ok, find saturation value at full brightness
scaledSatVal _ 255 - PressDefs.MulDiv[255-fullSatVal,currentSat,255];
--and move between that and black according to brightness
gray _ PressDefs.MulDiv[scaledSatVal,currentBr,255];
currentGray _ (gray+1)/16;
GraphicsDefs.SetGrayLevel[currentGray];
END;

DisplayPressPage  : PUBLIC PROCEDURE [p: POINTER TO PressDefs.PressPage] =
BEGIN OPEN PressDefs;
currentDLRecord: CARDINAL _ 177777B;
spaceX: CARDINAL _ 177777B;
spaceY: CARDINAL _ 0;
obj: POINTER TO PressObject _ p.ObjectList;
micaX,micaY: CARDINAL;
chStr: STRING _ "a";
currentFontIndex: CARDINAL _ 177777B;
currentFont: POINTER TO GraphicsDefs.StrikeFont;
currentMode: GraphicsDefs.textMode;
currentWidth: POINTER TO ARRAY CHARACTER [0C..255C] OF CARDINAL;
currentSat,currentHue,currentBr,currentGray: CARDINAL _ 0;
AlternativeOn: BOOLEAN _ FALSE;

      DoChars : PROCEDURE[obj: POINTER TO PressDefs.PressObject,s: STRING] =
            BEGIN
            i: CARDINAL;
            micaWidth: CARDINAL _ 0;
            eBox: POINTER TO StrawDefs.EntityBox _
                  SystemDefs.AllocateHeapNode[SIZE[StrawDefs.EntityBox]];
            eBitmap: POINTER TO GraphicsDefs.Bitmap;

            FOR i IN [0..s.length) DO
                  IF s[i] = '  AND spaceX # 177777B THEN micaWidth _ micaWidth + spaceX
                  ELSE micaWidth _ micaWidth + currentWidth[s[i]];
            ENDLOOP;

            eBox^_[Box:,Object:obj];
            GraphicsDefs.CreateBox[@eBox.Box,
                  PressDefs.MulDiv[micaWidth,bitsPerInch,micasPerInch]+4,--4 is fudge
                  currentFont.ascent+currentFont.descent];
            eBitmap _ @eBox.Box.boxBitmap;
            micaWidth _ 0;
            FOR i IN [0..s.length) DO
                  chStr[0]_s[i];
                  IF s[i] = '  AND spaceX#177777B THEN
                              BEGIN
                              micaWidth _ micaWidth + spaceX;
                              LOOP;
                              END;
                  [] _ GraphicsDefs.PutText[chStr,
                              PressDefs.MulDiv[micaWidth,bitsPerInch,micasPerInch]+1,
                              currentFont.ascent,currentFont,currentMode,eBitmap];
                  micaWidth _ micaWidth+ currentWidth[s[i]];
            ENDLOOP;

            [eBox.Box.displayX,eBox.Box.displayY] _ DisplayCoord[micaX,micaY];
            micaX _ micaX + micaWidth;
            eBox.Box.function _ paint;
            IF currentGray # 0 THEN
                  BEGIN
                  eBox.Box.type _ gray;
                  eBox.Box.gray _ currentGray;
                  END;
```

```
            AddToDisplay[@eBox.Box,eBox.Box.displayX,eBox.Box.displayY-currentFont.ascent];
            END;

    GraphicsDefs.SetGrayLevel[0];
    UNTIL obj = NIL DO
        micaX _ obj.micaX;
        micaY _ obj.micaY;
        spaceX _ obj.spaceX;
        spaceY _ obj.spaceY;
        IF (currentBr#obj.br) OR (currentHue#obj.hue) OR (currentSat#obj.sat) THEN
            BEGIN
            currentGray _ SetGray[obj.hue,obj.sat,obj.br];
            currentBr _ obj.br;
            currentHue _ obj.hue;
            currentSat _ obj.sat;
            END;
    maxDataBytes _ MAX[obj.nDataBytes,maxDataBytes];
        SELECT obj.command FROM
            ELShowCharacters=>
                BEGIN
                string: STRING _ SystemDefs.AllocateHeapString[obj.nDataBytes];
                dp: DiscoDefs.DiskPosition _ obj.diskPosition;
                DiscoDefs.TransferBytes[dp: @dp,nBytes: obj.nDataBytes,
                    data: LOOPHOLE[@string.text]];
                string.length _ obj.nDataBytes;
                IF currentFontIndex # obj.font THEN
                    [currentFont,currentMode,currentWidth,currentFontIndex] _
                                                                    StrawDefs.GetFont[obj.font];
                DoChars[obj,string];
                SystemDefs.FreeHeapString[string];
                END;
            ELShowObject=>
                BEGIN
                eBox: POINTER TO StrawDefs.EntityBox _
                            SystemDefs.AllocateHeapNode[SIZE[StrawDefs.EntityBox]];
                eBox.Object _ obj;
                CreateObjectBox[eBox];
                AddToDisplay[@eBox.Box,eBox.Box.displayX,eBox.Box.displayY];
                END;
            ELShowDots,ELShowDotsOpaque=> --EL[i] tells which
                BEGIN
                do: POINTER TO StrawDefs.DotObject _
                            SystemDefs.AllocateHeapNode[SIZE[StrawDefs.DotObject]];
                do^ _ [Box:,Object: obj,DotsData: LOOPHOLE[obj+SIZE[PressObject]]];
                CreateDotsBox[do];
                GraphicsDefs.SetGrayLevel[currentGray]; --reset to prev value
                AddToDisplay[@do.Box,do.Box.displayX,do.Box.displayY];
                END;
            ELShowRectangle=>
                BEGIN
                x1,y1,x2,y2: CARDINAL;
                eBox: POINTER TO StrawDefs.EntityBox _
                            SystemDefs.AllocateHeapNode[SIZE[StrawDefs.EntityBox]];
                [x1,y2] _ StrawDefs.DisplayCoord[micaX,micaY];
                [x2,y1] _ StrawDefs.DisplayCoord[micaX+spaceX,micaY+spaceY];
                x2 _ PressDefs.MulDiv[spaceX,bitsPerInch,micasPerInch];
                y2 _ PressDefs.MulDiv[spaceY,bitsPerInch,micasPerInch];
                eBox^_[Box:,Object: obj];
                GraphicsDefs.CreateBox[@eBox.Box,x2+1,y2+1];
                GraphicsDefs.PutArea[0,0,x2,y2,@eBox.Box.boxBitmap];
                IF currentGray # 0 THEN
                    BEGIN
                    eBox.Box.type _ gray;
                    eBox.Box.gray _ currentGray;
                    END;
                AddToDisplay[@eBox.Box,x1,y1];
                END;
            ENDCASE=>NULL;
        obj _ obj.link;
    ENDLOOP; --until obj = NIL

    END; --DisplayPressPage
```

```
CreateDotsBox  : PUBLIC PROCEDURE [do: POINTER TO StrawDefs.DotObject] =
BEGIN
dispX,dispY: CARDINAL;
pdd: POINTER TO PressDefs.PressDotsData _ do.DotsData;
obj: POINTER TO PressDefs.PressObject _ do.Object;

dispX _ PressDefs.MulDiv[pdd.micaWidth,bitsPerInch,micasPerInch];
dispY _ PressDefs.MulDiv[pdd.micaHeight,bitsPerInch,micasPerInch];
GraphicsDefs.CreateBox[@do.Box,dispX,dispY];
IF pdd.nBitsPerPixel = 0 THEN
    BEGIN
    nWords: CARDINAL = (pdd.nPixels/16)*pdd.nLines;
    nChunks: CARDINAL = (nWords+4095)/4096;
    sArea: GraphicsDefs.Rectangle_[x1:0,y1:0,x2:pdd.displayPixels-1,y2:pdd.nLines/nChunks-1];
    dArea: GraphicsDefs.Rectangle _ [x1:0,y1:0,x2:dispX-1,y2:];
    temp: GraphicsDefs.Bitmap _
     [  bank:0,nWords:pdd.nPixels/16,nBits:pdd.nPixels,nLines:
        pdd.nLines/nChunks,bits:
        SystemDefs.AllocateSegment[pdd.nPixels/16+nWords/nChunks]];
    dataPos: DiscoDefs.DiskPosition _ pdd.diskPosition;
    i: CARDINAL;
    FOR i IN [1..nChunks] DO
        sArea.y2 _ (pdd.nLines*i)/nChunks-(pdd.nLines*(i-1))/nChunks-1;
        dArea.y2 _ (dispY*i)/nChunks-1;
        DiscoDefs.TransferBytes[dp:@dataPos,nBytes:
            ((pdd.nLines*i)/nChunks-(pdd.nLines*(i-1))/nChunks)*(pdd.nPixels/8),
            data:LOOPHOLE[temp.bits]];
        GraphicsDefs.TransferRectangle[src:@temp,dest:@do.Box.boxBitmap,
            srcRectangle:@sArea,destRectangle:@dArea];
        dArea.y1 _ dArea.y2+1;
    ENDLOOP;
    SystemDefs.FreeSegment[temp.bits];
    END
ELSE
    BEGIN
    IF pdd.nBitsPerPixel = 8 AND pdd.fileName = NIL THEN
        DoHalftone[pdd,@do.Box.boxBitmap]
    ELSE -- no can do
        BEGIN
        GraphicsDefs.SetGrayLevel[14];
        GraphicsDefs.PutGray[0,0,dispX,dispY,@do.Box.boxBitmap];
        END;
    END;
do.Box.function _ IF pdd.opaque THEN replace ELSE paint;
[do.Box.displayX,do.Box.displayY] _
    StrawDefs.DisplayCoord[obj.micaX,obj.micaY+pdd.micaHeight];
--kludge: got to store mica width, height
do.Object.spaceX _ pdd.micaWidth;
do.Object.spaceY _ pdd.micaHeight;
END; --end of ShowDots

CreateObjectBox  : PUBLIC PROCEDURE [eBox: POINTER TO StrawDefs.EntityBox] =
BEGIN
obj: POINTER TO PressDefs.PressObject _ eBox.Object;
Box: POINTER TO GraphicsDefs.Box _ @eBox.Box;
cX,cY,x1,y1,x2,y2,j: CARDINAL;
ObjCommand: TYPE = RECORD
    [move:BOOLEAN,x,y: CARDINAL];
maxObj: CARDINAL =  maxObjCommandSize  /SIZE[ObjCommand];
ObjList: ARRAY [0..maxObj) OF ObjCommand;
nObjects: CARDINAL _ 0;
objCommand: POINTER TO ARRAY [0..0) OF CARDINAL _
    SystemDefs.AllocateSegment[(obj.nDataBytes+1)/2];
minX,minY: CARDINAL _ 177777B;
maxX,maxY: CARDINAL _ 0;
currentX,currentY: CARDINAL;
commIndex: CARDINAL _ 0;
dp: DiscoDefs.DiskPosition _ obj.diskPosition;
DiscoDefs.TransferBytes[dp: @dp,nBytes: obj.nDataBytes,
    data: LOOPHOLE[objCommand]];
UNTIL commIndex >= obj.nDataBytes/2 DO
```

```
SELECT objCommand[commIndex] FROM
    PressDefs.DLMoveTo =>
        BEGIN
        currentX _ obj.micaX + objCommand[commIndex+1];
        currentY _ obj.micaY + objCommand[commIndex+2];
        commIndex _ commIndex+3;
        ObjList[nObjects] _ [TRUE,currentX,currentY];
        minX _ MIN[minX,currentX];maxX _ MAX[maxX,currentX];
        minY _ MIN[minY,currentY];maxY _ MAX[maxY,currentY];
        nObjects _ nObjects+1;
        IF nObjects > maxObj THEN MiscDefs.CallDebugger["obj overflow"];
        END;
    PressDefs.DLDrawTo =>
        BEGIN
        currentX _ obj.micaX + objCommand[commIndex+1];
        currentY _ obj.micaY + objCommand[commIndex+2];
        commIndex _ commIndex+3;
        ObjList[nObjects] _ [FALSE,currentX,currentY];
        minX _ MIN[minX,currentX];maxX _ MAX[maxX,currentX];
        minY _ MIN[minY,currentY];maxY _ MAX[maxY,currentY];
        nObjects _ nObjects+1;
        IF nObjects > maxObj THEN MiscDefs.CallDebugger["obj overflow"];
        END;
    PressDefs.DLDrawCurve =>
        BEGIN
        fake: ARRAY[0..2) OF INTEGER;
        coeff: ARRAY[0..6) OF INTEGER;
        dx,dy: ARRAY[0..3) OF INTEGER;
        commIndex _ commIndex+1;
        FOR j IN [0..6) DO
            fake[0]_objCommand[commIndex];
            fake[1]_objCommand[commIndex+1];
            commIndex _ commIndex+2;
            coeff[j] _ RealConvert[fake];
        ENDLOOP;
        ControlPoints[@coeff,@dx,@dy];
        FOR j IN [KLUDGESegs ..3) DO
            cX _ currentX + dx[j];cY _ currentY + dy[j];
            ObjList[nObjects] _ [FALSE,cX,cY];
            minX _ MIN[minX,cX];maxX _ MAX[maxX,cX];
            minY _ MIN[minY,cY];maxY _ MAX[maxY,cY];
            nObjects _ nObjects+1;
            IF nObjects > maxObj THEN MiscDefs.CallDebugger["obj overflow"];
        ENDLOOP;
        currentX _ cX;currentY _ cY;
        END;
    ENDCASE => MiscDefs.CallDebugger["bad object"];
ENDLOOP;

SystemDefs.FreeSegment[objCommand];
--now, process linked list of commands
[x1,y2] _ StrawDefs.DisplayCoord[minX,minY];
[x2,y1] _ StrawDefs.DisplayCoord[maxX,maxY];
GraphicsDefs.CreateBox[Box,x2+1-x1,y2+1-y1];
FOR j IN [0..nObjects) DO
    [cX,cY] _ StrawDefs.DisplayCoord[ObjList[j].x,ObjList[j].y];
    cX _ cX-x1;cY _ cY-y1;
    IF NOT ObjList[j].move THEN
        GraphicsDefs.PutLine[currentX,currentY,cX,cY,
                @Box.boxBitmap];
    currentX _ cX;currentY _ cY;
ENDLOOP;
[cX,cY] _ StrawDefs.DisplayCoord[ObjList[0].x,ObjList[0].y];
GraphicsDefs.PutLine[currentX,currentY,cX-x1,cY-y1,@Box.boxBitmap];
Box.function _ paint;
Box.displayX _ x1;Box.displayY _ y1;
--kludge: got to store mica offset from position to corner
BEGIN
e: POINTER TO  StrawDefs.EntityBox _ LOOPHOLE[Box];
e.Object.spaceX _ minX-e.Object.micaX;
e.Object.spaceY _ maxY-e.Object.micaY;
END;
```

```
END;

AddToDisplay  : PROCEDURE [box: POINTER TO GraphicsDefs.Box,x,y: CARDINAL] =
BEGIN
DisplayListDefs.AddDisplayItem[box];
box.displayX _ x;
box.displayY _ y;
box.displayBitmap _ Screen;
box.displayed _ TRUE;
SELECT box.type FROM
      gray =>GraphicsDefs.PutGrayBitmap[@box.boxBitmap,x,y];
      normal =>SELECT box.function FROM
            replace =>GraphicsDefs.ReplaceBitmap[@box.boxBitmap,x,y];
            paint =>GraphicsDefs.PutBitmap[@box.boxBitmap,x,y];
            ENDCASE => ERROR;
      ENDCASE => ERROR;
END;

DisplayCoord  : PUBLIC PROCEDURE [x,y: INTEGER] RETURNS [dispX,dispY: INTEGER] =
BEGIN
dispX _ dispXOff + PressDefs.MulDiv[x,bitsPerInch,micasPerInch];
dispY _ MAX[0,
                  INTEGER[Screen.nLines-(dispYOff + PressDefs.MulDiv[y,bitsPerInch,micasPerInch])]];
END;

PageCoord  : PUBLIC PROCEDURE [dispX,dispY: INTEGER] RETURNS [x,y: INTEGER] =
BEGIN
x _ PressDefs.MulDiv[dispX-dispXOff,micasPerInch,bitsPerInch];
y _ PressDefs.MulDiv[Screen.nLines - (dispY-dispYOff),micasPerInch,bitsPerInch];
END;

LongDiv32  : PROCEDURE[CARDINAL,CARDINAL,CARDINAL] RETURNS[CARDINAL] =
MACHINE CODE BEGIN Mopcodes.zLDIV;END;
--LongDiv32  : PROCEDURE[ARRAY [0..1] OF CARDINAL,CARDINAL] RETURNS[CARDINAL] =MACHINE CODE BEGIN
Mopcodes.zEXCH;Mopcodes.zLDIV;END;

END.
```

```
DIRECTORY
    DisplayListDefs: FROM "DisplayListDefs",
    GraphicsDefs: FROM "GraphicsDefs",
    InlineDefs,
    IODefs: FROM "IODefs" USING [GetInputStream],
    MagicDefs: FROM "MagicDefs" USING [Menu],
    PressDefs: FROM "PressDefs" USING [PressObject,SignedMulDiv,MulDiv,
        ELShowRectangle,ELShowObject,ELShowCharacters,ELShowDots,
        ELShowDotsOpaque,PressDotsData],
    ProcessDefs: FROM "ProcessDefs"  USING [Yield],
    StrawDefs: FROM "StrawDefs",
    StreamDefs: FROM "StreamDefs" USING [StreamHandle],
    SystemDefs: FROM "SystemDefs" USING
        [AllocateHeapNode,FreeHeapNode];

StrawSil : PROGRAM IMPORTS DisplayListDefs,GraphicsDefs,InlineDefs,IODefs,MagicDefs,
PressDefs,ProcessDefs,StrawDefs,SystemDefs
    EXPORTS StrawDefs =
BEGIN
Buttons : TYPE = MACHINE DEPENDENT RECORD
    [ KeyPadAndGarbage: [0..10000B),
      Mark: BOOLEAN, --TRUE means button up
      Select: BOOLEAN,
      Draw: BOOLEAN
    ];
KeyWord2  : TYPE = MACHINE DEPENDENT RECORD
    [ One,Esc,Tab,F,Ctrl,C,J,B,Z,LShift,Period,Semi,Return,LArrow,Del,xxx: BOOLEAN
    ];
MouseButtons  : POINTER TO Buttons = LOOPHOLE[177030B];
KeyWord : POINTER TO KeyWord2 = LOOPHOLE[177036B];
MouseX : POINTER TO CARDINAL = LOOPHOLE[424B];
MouseY : POINTER TO CARDINAL = LOOPHOLE[425B];
CursorX  : POINTER TO CARDINAL = LOOPHOLE[426B];
CursorY  : POINTER TO CARDINAL = LOOPHOLE[427B];
bitsPerInch  : CARDINAL _ 72;
micasPerInch  : CARDINAL = 2540;
keys: StreamDefs.StreamHandle _ IODefs.GetInputStream[];
displayList: POINTER TO ARRAY [0..0) OF POINTER TO GraphicsDefs.Box;
selectionIndex: CARDINAL;
selectGray: CARDINAL _ 15;
waitCount: CARDINAL _ 40;
UnDeleteRecord  : TYPE = RECORD
 [    count: CARDINAL,
      item: ARRAY [0..0) OF POINTER TO GraphicsDefs.Box
 ];
RealUnDeleteMax  : CARDINAL = 6;
UnDeleteStack  : ARRAY [1..RealUnDeleteMax] OF POINTER TO UnDeleteRecord;
UnDeleteStackPointer  : CARDINAL _ 0;
UnDeleteStackMax  : CARDINAL _ RealUnDeleteMax-1;
Ctrl: ARRAY CHARACTER ['A..'Z] OF CHARACTER =
    [1C,2C,3C,4C,5C,6C,7C,10C,11C,12C,13C,
        14C,15C,16C,17C,20C,21C,22C,23C,24C,25C,26C,27C,30C,31C,32C];
ScreenHeight  : CARDINAL _ GraphicsDefs.GetDefaultBitmapHandle[].nLines;

Mica : TYPE = CARDINAL;
OriginX: Mica _ 0;
OriginY: Mica _ 0;
MarkX: Mica _ 0;
MarkY: Mica _ 0;
ButtonX: Mica _ 0;
ButtonY: Mica _ 0;

dc: PROCEDURE [m: Mica] RETURNS [CARDINAL] = INLINE
BEGIN
RETURN[PressDefs.SignedMulDiv[m,bitsPerInch,micasPerInch]];
END;

mc: PROCEDURE [c: CARDINAL] RETURNS [Mica] = INLINE
BEGIN
RETURN[PressDefs.MulDiv[c,micasPerInch,bitsPerInch]];
END;
```

```
EditDisplayList  : PUBLIC PROCEDURE =
BEGIN OPEN DisplayListDefs;
i: CARDINAL;
ButtonState: Buttons;
MagnifyOn: BOOLEAN _ FALSE;
FlashMarks: PROCEDURE =
      BEGIN
      GraphicsDefs.XorArea[dc[OriginX],dc[OriginY],dc[OriginX]+3,dc[OriginY]+1];
      GraphicsDefs.XorArea[dc[MarkX],dc[MarkY],dc[MarkX]+1,dc[MarkY]+6];
      FOR i IN [0..waitCount] DO ProcessDefs.Yield[];ENDLOOP;
      GraphicsDefs.XorArea[dc[OriginX],dc[OriginY],dc[OriginX]+3,dc[OriginY]+1];
      GraphicsDefs.XorArea[dc[MarkX],dc[MarkY],dc[MarkX]+1,dc[MarkY]+6];
      END;

displayList _ DisplayListDefs.GetDisplayListHandle[];
selectionIndex _ DisplayListDefs.GetDisplayCount[];
OriginX _ 0;
OriginY _ 0;

DO --edit loop
      DO --wait for all buttons to be released
      ButtonState _ MouseButtons^;
      IF (ButtonState.Mark AND ButtonState.Select AND ButtonState.Draw) OR
            (NOT keys.endof[keys])
                  THEN EXIT;
      FlashMarks[];
      ENDLOOP;
      DO --wait for a button to go down
      ButtonState _ MouseButtons^;
      IF (NOT (ButtonState.Mark AND ButtonState.Select AND ButtonState.Draw)) OR
            (NOT keys.endof[keys])
                  THEN EXIT;
      --check for cursor inside top selected image
      i _ DisplayListDefs.GetDisplayCount[]-1;
      IF i >= selectionIndex AND (NOT MagnifyOn) AND
            DisplayListDefs.Inside[CursorX^,CursorY^,displayList[i]] THEN
            BEGIN OPEN PressDefs;
            d: POINTER TO Straw Defs.DotObject    _ LOOPHOLE[displayList[i]];
            IF d.Object.command IN [ELShowDots..ELShowDotsOpaque] AND
                  d.DotsData.nBitsPerPixel = 8 THEN StrawDefs.EditGrayImage[d];
            END;
      FlashMarks[];
      ENDLOOP;

      IF NOT keys.endof[keys] THEN
            BEGIN
            ch: CHARACTER _ keys.get[keys];
            IF MagnifyOn THEN StrawDefs.StopMagnifier[];
            SELECT ch FROM
                  Ctrl['C] =>
                        BEGIN
                        Copy[MarkX-OriginX,MarkY-OriginY];
                        OriginX _ MarkX;OriginY _ MarkY;
                        END;
                  Ctrl['D] => Delete[];
                  Ctrl['I] =>
                        BEGIN
                        haveSelection: BOOLEAN _
                              selectionIndex<DisplayListDefs.GetDisplayCount[];
                        IF KeyWord.Ctrl THEN LOOP; --was <TAB>
                        StrawDefs.SelectFile[];
                        IF haveSelection THEN
                              BEGIN UnDeleteStackMax _ UnDeleteStackMax+1;Delete[];
                              END;
                        StrawDefs.AddFile[];
                        selectionIndex _ DisplayListDefs.GetDisplayCount[];
                        IF haveSelection THEN
                              BEGIN UnDelete[];UnDeleteStackMax _ UnDeleteStackMax-1;
                              END;
                        END;
                  Ctrl['K] =>
                        BEGIN
```

```
                ClearUnDeleteStack[];
                StrawDefs.NewPage[];
                StrawDefs.Refresh[];
                selectionIndex _ 0;
                END;
            Ctrl['M] =>
                BEGIN
                IF KeyWord.Ctrl THEN LOOP; --was <CR>
                MagnifyOn _ NOT MagnifyOn;
                IF MagnifyOn THEN
                    BEGIN
                    StrawDefs.SetMagnifierPosition[MarkX,MarkY];
                    END;
                END;
            Ctrl['O] => StrawDefs.WriteFile[];
            Ctrl['Q] => EXIT;
            Ctrl['T] =>
                BEGIN nItems: CARDINAL _ DisplayListDefs.GetDisplayCount[];
                RemoveSelections[];ResetDisplayCount[selectionIndex];
                StrawDefs.SetGrid[];
                ResetDisplayCount[nItems];ShowSelections[];
                END;
            Ctrl['U] => BEGIN NewSelection[];UnDelete[];END;
            Ctrl['X] =>
                BEGIN
                temp: CARDINAL;
                Move[MarkX-OriginX,MarkY-OriginY];
                temp _ MarkX;MarkX _ OriginX;OriginX _ temp;
                temp _ MarkY;MarkY _ OriginY;OriginY _ temp;
                END;
            ENDCASE;
        IF MagnifyOn THEN StrawDefs.StartMagnifier[selectionIndex,selectGray];
        LOOP;
        END;

IF MagnifyOn THEN [ButtonX,ButtonY] _
    StrawDefs.MagToMicaCoords[CursorX^,CursorY^]
ELSE BEGIN ButtonX _ mc[CursorX^];ButtonY _ mc[CursorY^];END;

IF NOT ButtonState.Select THEN
    BEGIN
    IF MagnifyOn THEN StrawDefs.StopMagnifier[];
    IF NOT (KeyWord.Ctrl OR KeyWord.LShift) THEN --"DeSelect"
        BEGIN
        DeSelect[];
        END
    ELSE IF NOT KeyWord.Ctrl THEN --"AddSelect"
        BEGIN
        Select[];
        END
    ELSE IF NOT KeyWord.LShift THEN --"AreaSelect"
        BEGIN
        OriginX _ ButtonX;
        OriginY _ ButtonY;
        NewSelection[];
        SelectArea[];
        END
    ELSE
        BEGIN
        --"Select"
        NewSelection[];
        Select[];
        END;
    IF MagnifyOn THEN StrawDefs.StartMagnifier[selectionIndex,selectGray];
    LOOP;
    END;

IF NOT ButtonState.Draw THEN
    BEGIN
    IF MagnifyOn THEN StrawDefs.StopMagnifier[];
    IF NOT (KeyWord.Ctrl OR KeyWord.LShift) THEN --"UnDelete"
        BEGIN
```

```
                NewSelection[];
                UnDelete[];
                END
        ELSE IF NOT KeyWord.Ctrl THEN --"Copy"
                BEGIN
                dx: INTEGER _ ButtonX - OriginX;
                dy: INTEGER _ ButtonY - OriginY;
                Copy[dx,dy];
                MarkX _ OriginX _ OriginX + dx;
                MarkY _ OriginY _ OriginY + dy;
                END
        ELSE IF NOT KeyWord.LShift THEN --"Delete"
                BEGIN
                NewSelection[];
                Select[];
                Delete[];
                END
        ELSE
                BEGIN
                --"Help"
                Help[];
                END;
        IF MagnifyOn THEN StrawDefs.StartMagnifier[selectionIndex,selectGray];
        LOOP;
        END;

IF NOT ButtonState.Mark THEN
        BEGIN
        IF NOT (KeyWord.Ctrl OR KeyWord.LShift) THEN --"Move Magnifier"
                BEGIN
                IF NOT MagnifyOn THEN
                        BEGIN
                        MagnifyOn _ TRUE;
                        StrawDefs.SetMagnifierPosition[ButtonX,ButtonY];
                        StrawDefs.StartMagnifier[selectionIndex,selectGray];
                        END;
                WHILE NOT (KeyWord.Ctrl OR KeyWord.LShift OR MouseButtons.Mark)
                DO
                        [ButtonX,ButtonY] _
                                StrawDefs.MagToMicaCoords[CursorX^,CursorY^];
                        StrawDefs.SetMagnifierPosition[ButtonX,ButtonY];
                        StrawDefs.ShowMagnifier[];
                ENDLOOP;
                END
        ELSE
                BEGIN
                IF MagnifyOn THEN
                        IF StrawDefs.ModifyMagnifier[dc[ButtonX],dc[ButtonY]] THEN
                                BEGIN
                                WHILE (NOT MouseButtons.Mark) AND
                                        StrawDefs.ModifyMagnifier[CursorX^,CursorY^] DO ENDLOOP;
                                LOOP;
                                END
                        ELSE StrawDefs.StopMagnifier[];
                IF NOT KeyWord.Ctrl THEN --Move
                        BEGIN
                        dx: INTEGER _ ButtonX - OriginX;
                        dy: INTEGER _ ButtonY - OriginY;
                        Move[dx,dy];
                        MarkX _ OriginX;
                        MarkY _ OriginY;
                        OriginX _ OriginX + dx;
                        OriginY _ OriginY + dy;
                        END
                ELSE IF NOT KeyWord.LShift THEN --"MoveOrigin"
                        BEGIN
                        OriginX _ ButtonX;
                        OriginY _ ButtonY;
                        END
                ELSE
                        BEGIN
                        --"Mark"
```

```
                    MarkX _ ButtonX;
                    MarkY _ ButtonY;
                    END;
                IF MagnifyOn THEN StrawDefs.StartMagnifier[selectionIndex,selectGray];
                END; --non-magnify stuff
                LOOP;
            END;

ENDLOOP; --main loop
RemoveSelections[];
FOR i IN [selectionIndex..DisplayListDefs.GetDisplayCount[]) DO UnSelect[i]; ENDLOOP;
--and clear out the UnDeleteStack
ClearUnDeleteStack[];
END;
Help: PROCEDURE =
BEGIN
HelpMenu: ARRAY [0..7) OF STRING _
 [      "      |       Shift  Ctrl  CtlShft",
        "-----------------------------------",
        "Mark  | MARK ORIGIN  MOVE  MAGNIFY",
        "Draw  | HELP DELETE  COPY  UNDELETE",
        "Select| SEL  AREA    ADD   DESELECT",
        "-----------------------------------",
        "  keys| ^C,^D,^I,^K,^M,^O,^Q,^T,^U,^X"
 ];
font: POINTER TO GraphicsDefs.StrikeFont _ GraphicsDefs.GetStrikeHandle["Gacha10"];
[] _ MagicDefs.Menu[7,LOOPHOLE[@HelpMenu],font];
END;

Copy: PROCEDURE [dx,dy: Mica] =
BEGIN
i: CARDINAL;
newEBox,eBox: POINTER TO StrawDefs.EntityBox;
obj: POINTER TO PressDefs.PressObject;
nItems: CARDINAL _ DisplayListDefs.GetDisplayCount[];
objSize: CARDINAL;
RemoveSelections[];
FOR i IN [selectionIndex..nItems) DO
        eBox _ LOOPHOLE[displayList[i]];
        IF eBox.Object.command IN
                [PressDefs.ELShowDots..PressDefs.ELShowDotsOpaque] THEN
                BEGIN
                do: POINTER TO StrawDefs.DotObject _
                        SystemDefs.AllocateHeapNode[SIZE[StrawDefs.DotObject]];
                newEBox _ LOOPHOLE[do];
                objSize _ SIZE[PressDefs.PressObject]+SIZE[PressDefs.PressDotsData];
                obj _ SystemDefs.AllocateHeapNode[objSize];
                do.DotsData _ LOOPHOLE[obj+SIZE[PressDefs.PressObject]];
                END
        ELSE
                BEGIN
                newEBox _ SystemDefs.AllocateHeapNode[SIZE[StrawDefs.EntityBox]];
                objSize _ SIZE[PressDefs.PressObject];
                obj _ SystemDefs.AllocateHeapNode[objSize];
                END;
        GraphicsDefs.CreateBox[@newEBox.Box,eBox.Box.boxBitmap.nBits,
                eBox.Box.boxBitmap.nLines];
        InlineDefs.COPY[from:eBox.Object,nwords:objSize,to:obj];
        newEBox.Object _ obj;
        newEBox.Box.function _ eBox.Box.function;
        newEBox.Box.type _ eBox.Box.type;
        newEBox.Box.gray _ eBox.Box.gray;

        newEBox.Object.micaX _ eBox.Object.micaX + dx;
        newEBox.Object.micaY _ eBox.Object.micaY - dy;
        GraphicsDefs.PutBitmap[@eBox.Box.boxBitmap,0,0,
                @newEBox.Box.boxBitmap];
        DisplayListDefs.AddDisplayItem[@newEBox.Box];
        [newEBox.Box.displayX,newEBox.Box.displayY] _
                ScreenCoords[newEBox];
ENDLOOP;
FOR i IN [selectionIndex..nItems) DO UnSelect[i];ENDLOOP;
```

```
ShowSelections[];
END;

RemoveSelections  : PROCEDURE =
BEGIN
i: CARDINAL;
FOR i DECREASING IN [selectionIndex..DisplayListDefs.GetDisplayCount[])
        DO GraphicsDefs.RemoveBox[displayList[i]]; ENDLOOP;
END;

ShowSelections  : PROCEDURE =
BEGIN
i: CARDINAL;
d: POINTER TO GraphicsDefs.Box;
FOR i IN [selectionIndex..DisplayListDefs.GetDisplayCount[]) DO
    d _ displayList[i];
    GraphicsDefs.DisplayBox[d,d.displayX,d.displayY];
ENDLOOP;
END;

ScreenCoords  : PROCEDURE [e: POINTER TO StrawDefs.EntityBox] RETURNS [x,y: CARDINAL] =
BEGIN OPEN PressDefs;
obj: POINTER TO PressObject _ e.Object;
--displayX,displayY refer to top left corner.
--different objects use different reference points, so be careful
SELECT e.Object.command FROM
    ELShowObject =>
        BEGIN
        x _ dc[obj.micaX+obj.spaceX];
        y _ ScreenHeight-dc[obj.micaY+obj.spaceY];
        END;
    ELShowRectangle,ELShowDots,ELShowDotsOpaque =>
        BEGIN
        x _ dc[obj.micaX];
        y _ ScreenHeight-dc[obj.micaY+obj.spaceY];
        END;
    ELShowCharacters =>
        BEGIN
        strike: POINTER TO GraphicsDefs.StrikeFont;
        [strike,,,] _ StrawDefs.GetFont[obj.font];
        x _ dc[obj.micaX];
        y _ ScreenHeight-(dc[obj.micaY]+strike.ascent);
        END;
    ENDCASE => ERROR; --can't move guys with unknown reference points
END;

Move : PROCEDURE [dx,dy: Mica] =
BEGIN
i: CARDINAL;
eBox: POINTER TO StrawDefs.EntityBox;
x,y: CARDINAL;
IF dx = 0 AND dy = 0 THEN RETURN;

IF (DisplayListDefs.GetDisplayCount[]-selectionIndex) > 1 THEN
    FOR i DECREASING IN [selectionIndex..DisplayListDefs.GetDisplayCount[])
        DO GraphicsDefs.RemoveBox[displayList[i]]; ENDLOOP;

FOR i IN [selectionIndex..DisplayListDefs.GetDisplayCount[])
DO
    eBox _ LOOPHOLE[displayList[i]];
    eBox.Object.micaX _ eBox.Object.micaX + dx;
    eBox.Object.micaY _ eBox.Object.micaY - dy;
    [x,y] _ ScreenCoords[eBox];
GraphicsDefs.MoveBox[@eBox.Box,x,y];
ENDLOOP;
END;

Delete : PROCEDURE =
BEGIN
i: CARDINAL;
nItems: CARDINAL _ DisplayListDefs.GetDisplayCount[];
d: POINTER TO GraphicsDefs.Box;
```

```
eBox: POINTER TO StrawDefs.EntityBox;
u: POINTER TO UnDeleteRecord;

IF selectionIndex = nItems THEN RETURN; --no action
IF UnDeleteStackPointer = UnDeleteStackMax THEN --real delete
      BEGIN
      FOR i IN [0..UnDeleteStack[1].count) DO
            d _ UnDeleteStack[1].item[i];
            eBox _ LOOPHOLE[d];
            GraphicsDefs.DestroyBox[d,FALSE];
            SystemDefs.FreeHeapNode[eBox.Object];
            SystemDefs.FreeHeapNode[d];
      ENDLOOP;
      SystemDefs.FreeHeapNode[UnDeleteStack[1]];
      FOR i IN [1..UnDeleteStackMax) DO
            UnDeleteStack[i] _ UnDeleteStack[i+1];
      ENDLOOP;
      END
ELSE UnDeleteStackPointer _ UnDeleteStackPointer+1;

u _ SystemDefs.AllocateHeapNode[(nItems-selectionIndex)+1];
u.count _ nItems-selectionIndex;
UnDeleteStack[UnDeleteStackPointer] _ u;

FOR i DECREASING IN [selectionIndex..nItems) DO
      d _ displayList[i];
      u.item[i-selectionIndex] _ d;
      DisplayListDefs.DeleteDisplayItem[d];
      StrawDefs.RestoreGrid[d];
ENDLOOP;
END;

UnDelete  : PROCEDURE =
BEGIN
i: CARDINAL;
u: POINTER TO UnDeleteRecord;
IF UnDeleteStackPointer = 0 THEN RETURN; --nothing left

u _ UnDeleteStack[UnDeleteStackPointer];
UnDeleteStackPointer _ UnDeleteStackPointer-1;
FOR i IN [0..u.count) DO
      DisplayListDefs.AddDisplayItem[u.item[i]];
ENDLOOP;
SystemDefs.FreeHeapNode[u];
ShowSelections[];
END;

ClearUnDeleteStack    : PROCEDURE =
BEGIN
d: POINTER TO GraphicsDefs.Box;
eBox: POINTER TO StrawDefs.EntityBox;
i: CARDINAL;
UNTIL UnDeleteStackPointer = 0 DO
      FOR i IN [0..UnDeleteStack[UnDeleteStackPointer].count) DO
            d _ UnDeleteStack[UnDeleteStackPointer].item[i];
            eBox _ LOOPHOLE[d];
            GraphicsDefs.DestroyBox[d,FALSE];
            SystemDefs.FreeHeapNode[eBox.Object];
            SystemDefs.FreeHeapNode[d];
      ENDLOOP;
SystemDefs.FreeHeapNode[UnDeleteStack[UnDeleteStackPointer]];
UnDeleteStackPointer _ UnDeleteStackPointer-1;
ENDLOOP;
END;

NewSelection  : PROCEDURE =
BEGIN OPEN GraphicsDefs;
d: POINTER TO Box;
i: CARDINAL;
RemoveSelections[];
FOR i IN [selectionIndex..DisplayListDefs.GetDisplayCount[]) DO
      d _ displayList[i];
```

```
        SetGrayLevel[selectGray];
        XorGray[0,0,d.boxBitmap.nBits,d.boxBitmap.nLines,@d.boxBitmap];
        DisplayBox[d,d.displayX,d.displayY];
        ReleaseBox[d];
ENDLOOP;
selectionIndex _ DisplayListDefs.GetDisplayCount[];
END;

Select : PROCEDURE =
BEGIN
i: CARDINAL;
x: CARDINAL _ dc[ButtonX];
y: CARDINAL _ dc[ButtonY];
FOR i DECREASING IN [0..selectionIndex) DO
        IF DisplayListDefs.Inside[x,y,displayList[i]] THEN
                BEGIN
                d: POINTER TO GraphicsDefs.Box _ displayList[i];
                RemoveSelections[];
                MakeSelect[i];
                ShowSelections[];
                EXIT;
                END;
ENDLOOP;
END;

SelectArea  : PROCEDURE =
BEGIN
i: CARDINAL;
x1: INTEGER _ dc[MIN[OriginX,MarkX]];
y1: INTEGER _ dc[MIN[OriginY,MarkY]];
x2: INTEGER _ dc[MAX[OriginX,MarkX]];
y2: INTEGER _ dc[MAX[OriginY,MarkY]];
d: POINTER TO GraphicsDefs.Box;
nItems: CARDINAL _ DisplayListDefs.GetDisplayCount[];
lastItem: CARDINAL _ nItems-1;

FOR i DECREASING IN [0..selectionIndex) DO
        d _ displayList[i];
        IF INTEGER[d.displayX] >= x1 AND INTEGER[d.displayY] >= y1 AND
                INTEGER[d.displayX+d.boxBitmap.nBits] <= x2 AND
                INTEGER[d.displayY+d.boxBitmap.nLines] <= y2 THEN
                        BEGIN
                        MakeSelect[i];
                        END;
ENDLOOP;

--first, need to unscramble them
FOR i IN [0..(nItems-selectionIndex)/2) DO
        d _ displayList[lastItem-i];
        displayList[lastItem-i] _ displayList[selectionIndex+i];
        displayList[selectionIndex+i] _ d;
ENDLOOP;

FOR i IN [selectionIndex..nItems) DO
        d _ displayList[i];
        GraphicsDefs.DisplayBox[d,d.displayX,d.displayY];
ENDLOOP;
OriginX _ MIN[OriginX,MarkX];
OriginY _ MIN[OriginY,MarkY];
END;

MakeSelect  : PROCEDURE [i: CARDINAL] =
BEGIN
d: POINTER TO GraphicsDefs.Box _ displayList[i];
e: POINTER TO StrawDefs.EntityBox _ LOOPHOLE[d];
DisplayListDefs.DeleteDisplayItem[d];
StrawDefs.RestoreGrid[d];
DisplayListDefs.AddDisplayItem[d];
GraphicsDefs.SetGrayLevel[selectGray];
GraphicsDefs.XorGray[0,0,d.boxBitmap.nBits,d.boxBitmap.nLines,@d.boxBitmap];
OriginX _ mc[d.displayX];
OriginY _ mc[d.displayY];
```

```
selectionIndex _ selectionIndex-1;
END;

DeSelect : PROCEDURE =
BEGIN
i: CARDINAL;
x: CARDINAL _ dc[ButtonX];
y: CARDINAL _ dc[ButtonY];
FOR i DECREASING IN [selectionIndex..DisplayListDefs.GetDisplayCount[]) DO
      IF DisplayListDefs.Inside[x,y,displayList[i]] THEN
            BEGIN
            RemoveSelections[];
            UnSelect[i];
            ShowSelections[];
            EXIT;
            END;
      ENDLOOP;
END;

UnSelect : PROCEDURE [i: CARDINAL] =
BEGIN
j: CARDINAL;
d: POINTER TO GraphicsDefs.Box _ displayList[i];
GraphicsDefs.SetGrayLevel[selectGray];
GraphicsDefs.XorGray[0,0,d.boxBitmap.nBits,d.boxBitmap.nLines,@d.boxBitmap];
GraphicsDefs.DisplayBox[d,d.displayX,d.displayY];
GraphicsDefs.ReleaseBox[d];
FOR j DECREASING IN [selectionIndex..i) DO
      displayList[j+1] _ displayList[j];
ENDLOOP;
displayList[selectionIndex] _ d;
selectionIndex _ selectionIndex+1;
END;


END.
```

```
DIRECTORY
    DiscoDefs: FROM "DiscoDefs",
    DisplayListDefs: FROM "DisplayListDefs",
    GraphicsDefs: FROM "GraphicsDefs",
    HalftoneDefs: FROM "HalftoneDefs",
    InlineDefs: FROM "InlineDefs" USING [COPY,LongMult,LongDiv],
    PressDefs: FROM "PressDefs" USING
        [MulDiv,SignedMulDiv,ELShowRectangle,ELShowDots,ELShowDotsOpaque,
        ELShowCharacters,ELShowObject,PressDotsData],
    StrawDefs: FROM "StrawDefs",
    SystemDefs: FROM "SystemDefs";

StrawMagnify : PROGRAM IMPORTS DiscoDefs,DisplayListDefs,GraphicsDefs,HalftoneDefs,InlineDefs,
PressDefs,StrawDefs,SystemDefs
    EXPORTS StrawDefs =
BEGIN
Mica: TYPE = CARDINAL;
Buttons: TYPE = MACHINE DEPENDENT RECORD
    [ KeyPadAndGarbage: [0..10000B),
      Mark: BOOLEAN, --TRUE means button up
      Select: BOOLEAN,
      Draw: BOOLEAN
    ];
MouseButtons: POINTER TO Buttons = LOOPHOLE[177030B];
CursorX: POINTER TO CARDINAL = LOOPHOLE[426B];
CursorY: POINTER TO CARDINAL = LOOPHOLE[427B];
bitsPerInch: CARDINAL _ 72;
micasPerInch: CARDINAL = 2540;

MagnifyWidth: CARDINAL _ 608/3;
MagnifyHeight: CARDINAL _ 808/3;
AdjustHeight: CARDINAL _ 12;
MagnifyBitsPerInch: CARDINAL _ 192;
AdjustX: CARDINAL _ ((MagnifyWidth-AdjustHeight)/6)*2;
MagnifyListCount: CARDINAL;
MagnifyListMax: CARDINAL _ 0;
MagnifyList: POINTER TO ARRAY[0..0) OF POINTER TO StrawDefs.EntityBox_NIL;
AdjustBox,MagnifyBox: GraphicsDefs.Box;
MagnifySelectionIndex: CARDINAL;
MagnifyX,MagnifyY: CARDINAL;
selectionIndex,selectGray: CARDINAL;
partitions: ARRAY [0..7) OF CARDINAL = [10,72,192,384,723,1270,2540];

MulDivCeiling : PROCEDURE [a,b,c:CARDINAL] RETURNS [CARDINAL] = INLINE
BEGIN
    al: LONG CARDINAL _ InlineDefs.LongMult[a,b];
    RETURN[InlineDefs.LongDiv[al+c-1,c]];
END;

StartMagnifier : PUBLIC PROCEDURE [Index,Gray: CARDINAL] =
BEGIN OPEN GraphicsDefs;
bm: POINTER TO GraphicsDefs.Bitmap _ @AdjustBox.boxBitmap;
AdjustXMin: CARDINAL _ MagnifyWidth-AdjustHeight;
i: CARDINAL;

selectionIndex _ Index;
selectGray _ Gray;
CreateBox[@MagnifyBox,MagnifyWidth+1,MagnifyHeight+1];
CreateBox[@AdjustBox,MagnifyWidth+1,AdjustHeight+1];
AdjustBox.displayBitmap _ MagnifyBox.displayBitmap _
                                        GraphicsDefs.GetDefaultBitmapHandle[];
EraseArea[0,0,MagnifyWidth,AdjustHeight,bm];
ReplaceArea[0,0,0,AdjustHeight,bm];
ReplaceArea[0,AdjustHeight,MagnifyWidth,AdjustHeight,bm];
ReplaceArea[MagnifyWidth,AdjustHeight,MagnifyWidth,0,bm];
ReplaceArea[MagnifyWidth,0,0,0,bm];
ReplaceArea[AdjustXMin,0,AdjustXMin,AdjustHeight,bm];

--icons
FOR i IN [0..6) DO
    ReplaceArea[(AdjustXMin*i)/6,AdjustHeight/2-i,
                    (AdjustXMin*(i+1))/6,AdjustHeight/2+i,bm];
```

```
ENDLOOP;

ReplaceArea[AdjustXMin+2,2,AdjustXMin+AdjustHeight-2,2,bm];
ReplaceArea[AdjustXMin+AdjustHeight-2,2,
                AdjustXMin+AdjustHeight-2,AdjustHeight-2,bm];
ReplaceArea[AdjustXMin+AdjustHeight-2,AdjustHeight-2,
                AdjustXMin+2,AdjustHeight-2,bm];
ReplaceArea[AdjustXMin+2,AdjustHeight-2,AdjustXMin+2,2,bm];

XorArea[AdjustX,1,AdjustX,AdjustHeight-1,bm];
ShowMagnifier[];
END;

ModifyMagnifier  : PUBLIC PROCEDURE [x,y: CARDINAL] RETURNS [BOOLEAN] =
BEGIN OPEN GraphicsDefs;
AdjustMin: CARDINAL _ AdjustBox.displayX;
AdjustMax: CARDINAL _ AdjustBox.displayX+MagnifyWidth-AdjustHeight;
X,Y,minX,minY: CARDINAL;
bm: POINTER TO Bitmap _ @AdjustBox.boxBitmap;

IF NOT DisplayListDefs.Inside[x,y,@AdjustBox] THEN RETURN[FALSE];
--change magnification
IF x IN [CARDINAL[MAX[0,INTEGER[AdjustBox.displayX]]]..AdjustMax] THEN
      BEGIN
      Len: CARDINAL _ AdjustMax-AdjustMin;
      p: CARDINAL _ ((x-AdjustMin)*6)/Len;
      thisPartStart: CARDINAL _ AdjustMin+(Len*p)/6;
      thisPartEnd: CARDINAL _ thisPartStart+Len/6;
      magnification: CARDINAL _ partitions[p] +
            PressDefs.MulDiv[x-thisPartStart,partitions[p+1]-partitions[p],Len/6];
      XorArea[AdjustX,1,AdjustX,AdjustHeight-1,bm];
      AdjustX _ x-AdjustMin;
      XorArea[AdjustX,1,AdjustX,AdjustHeight-1,bm];
      SetMagnification[magnification];
      ShowMagnifier[];
      RETURN [TRUE];
      END;
--change size
DO
      X _ MAX[16,INTEGER[(x-MagnifyX)*2]];
      Y _ MAX[16,INTEGER[(y-MagnifyY)*2]];
      minX _ MagnifyX - X/2;
      minY _ MagnifyY - Y/2;
      XorArea[minX,minY,minX+X,minY+1];
      XorArea[minX+X,minY,minX+X+1,minY+Y];
      XorArea[minX+X+1,minY+Y,minX,minY+Y];
      XorArea[minX,minY+Y+1,minX,minY];

      XorArea[minX,minY,minX+X,minY+1];
      XorArea[minX+X,minY,minX+X+1,minY+Y];
      XorArea[minX+X+1,minY+Y,minX,minY+Y];
      XorArea[minX,minY+Y+1,minX,minY];
      IF MouseButtons.Mark THEN EXIT;
      x _ CursorX^;
      y _ CursorY^;
ENDLOOP;
StopMagnifier[];
AdjustX _ PressDefs.MulDiv[AdjustX,X-AdjustHeight,MagnifyWidth-AdjustHeight];
SetMagnifierSize[X,Y];
StartMagnifier[selectionIndex,selectGray];
RETURN [TRUE];
END;

ShowMagnifier  : PUBLIC PROCEDURE =
BEGIN
i: CARDINAL;
--screen coords of source to be magnified
srcWidth: CARDINAL _ PressDefs.MulDiv[MagnifyWidth,bitsPerInch,MagnifyBitsPerInch];
srcHeight: CARDINAL _ PressDefs.MulDiv[MagnifyHeight,bitsPerInch,MagnifyBitsPerInch];
xMin: INTEGER _ MagnifyX - srcWidth/2;
xMax: INTEGER _ xMin + srcWidth;
yMin: INTEGER _ MagnifyY - srcHeight/2;
```

```
yMax: INTEGER _ yMin + srcHeight;
b: POINTER TO GraphicsDefs.Box;
e: POINTER TO StrawDefs.EntityBox;
srcRect,destRect,micaRect: GraphicsDefs.Rectangle;
bm: POINTER TO GraphicsDefs.Bitmap _ @MagnifyBox.boxBitmap;
displayList: POINTER TO ARRAY [0..0) OF POINTER TO GraphicsDefs.Box _
      DisplayListDefs.GetDisplayListHandle[];
screenHeight: INTEGER _ MagnifyBox.displayBitmap.nLines;
mx,my: INTEGER;
micaHeight,micaWidth,xLeft,yTop: INTEGER;
--precompute fixed values
micaXMin: INTEGER _ mc[MAX[-800,xMin]];
magXMin: INTEGER _ m[xMin];
micaYMin: INTEGER _ mc[screenHeight-MAX[0,yMin]];
magYMin: INTEGER _ m[yMin];
magScreenHeight: INTEGER _ m[screenHeight];
DefaultMagnify: PROCEDURE =
            BEGIN
            srcRect.x1 _ dc[micaRect.x1];
            srcRect.x2 _ dc[micaRect.x2];
            srcRect.y1 _ dc[micaRect.y1];
            srcRect.y2 _ dc[micaRect.y2];
            --there is a generous inclusion test, to allow accurate recomputing from mica
            --coords.  If the box is one line off the edge, the following error condition may occur
            IF srcRect.y2 < srcRect.y1 OR srcRect.x2 < srcRect.x1 THEN RETURN;
            destRect.x1 _ mx+mdc[micaRect.x1];
            destRect.x2 _ mx+mdc[micaRect.x2];
            destRect.y1 _ my+mdc[micaRect.y1];
            destRect.y2 _ my+mdc[micaRect.y2];
            IF (destRect.x1<0) OR (destRect.y1<0) THEN ERROR;
            IF i >= MagnifySelectionIndex THEN
                  BEGIN
                  GraphicsDefs.SetGrayLevel[selectGray];
                  GraphicsDefs.XorGray[0,0,bm.nBits,bm.nLines,bm];
                  END;
            SELECT b.type FROM
                  gray =>
                        BEGIN
                        GraphicsDefs.TransferRectangle[bm,@MagnifyBox.boxBitmap,
                              @srcRect,@destRect,erase];
                        GraphicsDefs.SetGrayLevel[b.gray];
                        GraphicsDefs.TransferRectangle[bm,@MagnifyBox.boxBitmap,
                              @srcRect,@destRect,paint,andgray];
                        END;
                  normal =>
                        GraphicsDefs.TransferRectangle[bm,@MagnifyBox.boxBitmap,
                              @srcRect,@destRect,b.function];
                  ENDCASE => ERROR;
            IF i >= MagnifySelectionIndex THEN
                  BEGIN
                  GraphicsDefs.SetGrayLevel[selectGray];
                  GraphicsDefs.XorGray[destRect.x1,destRect.y1,
                        destRect.x2,destRect.y2,@MagnifyBox.boxBitmap];
                  GraphicsDefs.XorGray[0,0,bm.nBits,bm.nLines,bm];
                  END;
            END; --default case

MagnifyListCount _ 0;
MagnifySelectionIndex _ selectionIndex;
GraphicsDefs.EraseArea[0,0,MagnifyWidth,MagnifyHeight,bm];

FOR i IN [0..DisplayListDefs.GetDisplayCount[]) DO
      b _ displayList[i];
      IF i = selectionIndex THEN MagnifySelectionIndex _ MagnifyListCount;
      IF INTEGER[b.displayX] > xMax OR INTEGER[b.displayY] > yMax OR
            INTEGER[b.displayX + b.boxBitmap.nBits] < xMin OR
            INTEGER[b.displayY + b.boxBitmap.nLines] < yMin THEN LOOP;
      e _ LOOPHOLE[b];
      IF MagnifyListCount >= MagnifyListMax THEN
            BEGIN
            newList: POINTER TO ARRAY [0..0) OF POINTER TO StrawDefs.EntityBox;
            newList _ SystemDefs.AllocateSegment[MagnifyListMax+256];
```

```
            InlineDefs.COPY[from:MagnifyList,nwords:MagnifyListMax,to:newList];
            IF MagnifyList # NIL THEN SystemDefs.FreeSegment[MagnifyList];
            MagnifyList _ newList;
            MagnifyListMax _ MagnifyListMax+256;
            END;
        MagnifyList[MagnifyListCount] _ e;
        MagnifyListCount _ MagnifyListCount+1;
    ENDLOOP;
    FOR i IN [0..MagnifyListCount) DO
        e _ MagnifyList[i];
        b _ @e.Box;
        bm _ @b.boxBitmap;

        SELECT e.Object.command FROM
            PressDefs.ELShowObject=>
                BEGIN
                micaWidth _ mc[e.Box.boxBitmap.nBits]-1;
                micaHeight _ mc[e.Box.boxBitmap.nLines]-1;
                xLeft _ e.Object.micaX+e.Object.spaceX;
                yTop _ e.Object.micaY+e.Object.spaceY;
                END;
            PressDefs.ELShowCharacters=>
                BEGIN
                strike: POINTER TO GraphicsDefs.StrikeFont;
                [strike,,,] _ StrawDefs.GetFont[e.Object.font];
                micaWidth _ mc[e.Box.boxBitmap.nBits]-1;
                micaHeight _ mc[e.Box.boxBitmap.nLines]-1;
                xLeft _ e.Object.micaX;
                yTop _ e.Object.micaY+mcr[strike.ascent];
                END;
            ENDCASE=> --ShowDots,ShowRectangle
                BEGIN
                micaWidth _ e.Object.spaceX;
                micaHeight _ e.Object.spaceY;
                xLeft _ e.Object.micaX;
                yTop _ e.Object.micaY+micaHeight;
                END;

        --max value of micaXMin,micaYMin is mc[-800]~-28822
        --so, overflow on add if nBits or nLines > (10/72)[32767-28222]~631
        mx _ micaXMin-xLeft;
        micaRect.x1 _ IF micaXMin < xLeft THEN 0 ELSE mx;
        mx _ mx+mmc[MagnifyBox.boxBitmap.nBits-1];
        micaRect.x2 _ IF mx < 0 THEN micaWidth ELSE MIN[micaWidth,mx];
        mx _ mdc[xLeft]-magXMin;
        IF mx < 0 THEN micaRect.x1 _ MulDivCeiling[-mx,micasPerInch,MagnifyBitsPerInch];
        my _ yTop - micaYMin;
        micaRect.y1 _ MAX[0,my];
        my _ my+mmc[MagnifyBox.boxBitmap.nLines-1];
        micaRect.y2 _ IF my < 0 THEN micaHeight ELSE MIN[micaHeight,my];
        my _ (magScreenHeight-mdc[yTop])-magYMin;
        IF my < 0 THEN micaRect.y1 _ MulDivCeiling[-my,micasPerInch,MagnifyBitsPerInch];

        SELECT e.Object.command FROM
        PressDefs.ELShowObject,
        PressDefs.ELShowCharacters =>
            DefaultMagnify[];
        PressDefs.ELShowRectangle =>
            BEGIN
            x1,y1,x2,y2: INTEGER;

            x1 _ mx+mdc[micaRect.x1];
            x2 _ mx+mdc[micaRect.x2];
            y1 _ my+mdc[micaRect.y1];
            y2 _ my+mdc[micaRect.y2];

            IF b.type = gray THEN
                BEGIN
                GraphicsDefs.SetGrayLevel[b.gray];
                GraphicsDefs.ReplaceGray[x1,y1,x2,y2,@MagnifyBox.boxBitmap];
                END
            ELSE GraphicsDefs.ReplaceArea[x1,y1,x2,y2,@MagnifyBox.boxBitmap];
```

```
        IF i >= MagnifySelectionIndex THEN
            BEGIN
            GraphicsDefs.SetGrayLevel[selectGray];
            GraphicsDefs.XorGray[x1,y1,x2,y2,@MagnifyBox.boxBitmap];
            END;
        END;
PressDefs.ELShowDots,PressDefs.ELShowDotsOpaque=>
        BEGIN
        do: POINTER TO StrawDefs.DotObject _ LOOPHOLE[e];
        pdd: POINTER TO PressDefs.PressDotsData _ do.DotsData;

        IF pdd.nBitsPerPixel # 8 OR pdd.fileName # NIL THEN
            BEGIN
            DefaultMagnify[];
            END
        ELSE
            BEGIN
            sampleVec: ARRAY [0..1000] OF CARDINAL;
            pixelRect: GraphicsDefs.Rectangle;
            bMode: CARDINAL _ pdd.mode/4;
            sMode: CARDINAL _ pdd.mode MOD 4;
            normal: BOOLEAN = sMode IN [2..3];
            xp,yp,xs,ys: CARDINAL;
            pmx: PROCEDURE [p: CARDINAL] RETURNS [m: CARDINAL] = INLINE
                BEGIN RETURN [PressDefs.MulDiv[p-xs,pdd.micaWidth,xp]];END;
            pmy: PROCEDURE [p: CARDINAL] RETURNS [m: CARDINAL] = INLINE
                BEGIN RETURN [PressDefs.MulDiv[p-ys,pdd.micaHeight,yp]];END;
            --Micas to Pixels
            mpx: PROCEDURE [m: CARDINAL] RETURNS [p: CARDINAL] = INLINE
                BEGIN RETURN [PressDefs.MulDiv[m,xp,pdd.micaWidth]+xs];END;
            mpy: PROCEDURE [m: CARDINAL] RETURNS [p: CARDINAL] = INLINE
                BEGIN RETURN [PressDefs.MulDiv[m,yp,pdd.micaHeight]+ys];END;
            --Round up Micas to Pixels
            rmpx: PROCEDURE [m: CARDINAL] RETURNS [p: CARDINAL] = INLINE
                BEGIN RETURN [MulDivCeiling[m,xp,pdd.micaWidth]+xs];END;
            rmpy: PROCEDURE [m: CARDINAL] RETURNS [p: CARDINAL] = INLINE
                BEGIN RETURN [MulDivCeiling[m,yp,pdd.micaHeight]+ys];END;

            IF normal THEN
                BEGIN
                xp _ pdd.displayPixels;yp _ pdd.displayLines;
                xs _ IF bMode IN [1..2]
                    THEN pdd.nPixels-(pdd.passPixels+pdd.displayPixels)
                    ELSE pdd.passPixels;
                ys _ IF sMode IN [1..2]
                    THEN pdd.nLines-(pdd.passLines+pdd.displayLines)
                    ELSE pdd.passLines;
                END
            ELSE
                BEGIN
                yp _ pdd.displayPixels;xp _ pdd.displayLines;
                ys _ IF bMode IN [1..2]
                    THEN pdd.nPixels-(pdd.passPixels+pdd.displayPixels)
                    ELSE pdd.passPixels;
                xs _ IF sMode IN [1..2]
                    THEN pdd.nLines-(pdd.passLines+pdd.displayLines)
                    ELSE pdd.passLines;
                END;

            --calculate SOURCE rectangle in pixels

            pixelRect.x1 _ rmpx[micaRect.x1]; --round up
            pixelRect.x2 _ mpx[micaRect.x2]; --don't include this pixel
            pixelRect.y1 _ rmpy[micaRect.y1]; --round up
            pixelRect.y2 _ mpy[micaRect.y2]; --don't include this pixel

            --and fix up mica rect to lie on real pixel boundaries
            micaRect.x1 _ pmx[pixelRect.x1]; --coord of left edge of pixel x1
            micaRect.x2 _ pmx[pixelRect.x2]; --coord of right edge of pixel x2-1
            micaRect.y1 _ pmy[pixelRect.y1]; --coord of top edge of pixel y1
            micaRect.y2 _ pmy[pixelRect.y2]; --coord of bottom edge of pixel y2-1
```

```
                destRect.x1 _ mx+mdc[micaRect.x1];
                destRect.x2 _ mx+mdc[micaRect.x2];
                destRect.y1 _ my+mdc[micaRect.y1];
                destRect.y2 _ my+mdc[micaRect.y2];
                IF (destRect.x1<0) OR (destRect.y1<0) THEN ERROR;

                IF pixelRect.x1 < pixelRect.x2 AND
                      pixelRect.y1 < pixelRect.y2 THEN
                      BEGIN
                      dp: DiscoDefs.DiskPosition _ pdd.diskPosition;
                      IF pdd.opaque THEN GraphicsDefs.EraseArea[destRect.x1,destRect.y1,
                            destRect.x2-1,destRect.y2-1,@MagnifyBox.boxBitmap];
                      IF normal THEN
                      HalftoneDefs.InitHalftone[destRect.x1,destRect.y1,
                            pixelRect.x2-pixelRect.x1,destRect.x2-destRect.x1,
                            pdd.min,pdd.max,@MagnifyBox.boxBitmap,
                            pixelRect.y2-pixelRect.y1,destRect.y2-destRect.y1,pdd.mode]
                      ELSE HalftoneDefs.InitHalftone[destRect.x1,destRect.y1,
                            pixelRect.y2-pixelRect.y1,destRect.y2-destRect.y1,
                            pdd.min,pdd.max,@MagnifyBox.boxBitmap,
                            pixelRect.x2-pixelRect.x1,destRect.x2-destRect.x1,pdd.mode];

                      SELECT sMode FROM
                            0=>THROUGH [0..pixelRect.x1) DO
                                  DiscoDefs.TransferBytes[dp:@dp,
                                        data:NIL,nBytes: pdd.nPixels];
                                  ENDLOOP;
                            1=>THROUGH [CARDINAL[pixelRect.x2]..pdd.nLines) DO
                                  DiscoDefs.TransferBytes[dp:@dp,
                                        data:NIL,nBytes: pdd.nPixels];
                                  ENDLOOP;
                            2=>THROUGH [CARDINAL[pixelRect.y2]..pdd.nLines) DO
                                  DiscoDefs.TransferBytes[dp:@dp,
                                        data:NIL,nBytes: pdd.nPixels];
                                  ENDLOOP;
                            ENDCASE=>THROUGH [0..pixelRect.y1) DO
                                  DiscoDefs.TransferBytes[dp:@dp,
                                        data:NIL,nBytes: pdd.nPixels];
                                  ENDLOOP;
                      DiscoDefs.TransferBytes[dp:@dp,
                            data:NIL,nBytes: SELECT bMode FROM
                            0=> pixelRect.x1,
                            1=> pdd.nPixels-pixelRect.x2,
                            2=> pdd.nPixels-pixelRect.y2,
                            ENDCASE=> pixelRect.y1];

                      IF normal THEN THROUGH [pixelRect.y1..pixelRect.y2) DO
                            DiscoDefs.TransferBytes[dp:@dp,
                                  data:LOOPHOLE[@sampleVec],nBytes: pdd.nPixels];
                            [] _ HalftoneDefs.PrintHalftoneLine[@sampleVec];
                      ENDLOOP
                      ELSE THROUGH [pixelRect.x1..pixelRect.x2) DO
                            DiscoDefs.TransferBytes[dp:@dp,
                                  data:LOOPHOLE[@sampleVec],nBytes: pdd.nPixels];
                            [] _ HalftoneDefs.PrintHalftoneLine[@sampleVec];
                      ENDLOOP;
                      END;
                      IF i >= MagnifySelectionIndex THEN
                            BEGIN
                            GraphicsDefs.SetGrayLevel[selectGray];
                            GraphicsDefs.XorGray[destRect.x1,destRect.y1,
                                  destRect.x2,destRect.y2,@MagnifyBox.boxBitmap];
                            END;
                END;
          END; --end of ShowDots
      ENDCASE => ERROR; --unknown display type encountered;
ENDLOOP;

bm _ @MagnifyBox.boxBitmap;
GraphicsDefs.ReplaceArea[0,0,0,MagnifyHeight,bm];
GraphicsDefs.ReplaceArea[0,MagnifyHeight,MagnifyWidth,MagnifyHeight,bm];
GraphicsDefs.ReplaceArea[MagnifyWidth,MagnifyHeight,MagnifyWidth,0,bm];
```

```
GraphicsDefs.ReplaceArea[MagnifyWidth,0,0,0,bm];
GraphicsDefs.RemoveBox[@AdjustBox];
GraphicsDefs.MoveBox[@MagnifyBox,MagnifyX-MagnifyWidth/2,
      MagnifyY-MagnifyHeight/2];
GraphicsDefs.DisplayBox[@AdjustBox,MagnifyX-MagnifyWidth/2,
      MIN[AdjustBox.displayBitmap.nLines-AdjustBox.boxBitmap.nLines-1,
                                                MagnifyY+MagnifyHeight/2]];
END;

StopMagnifier  : PUBLIC PROCEDURE =
BEGIN
GraphicsDefs.RemoveBox[@AdjustBox];
GraphicsDefs.ReleaseBox[@AdjustBox];
GraphicsDefs.DestroyBox[@AdjustBox,FALSE];
GraphicsDefs.RemoveBox[@MagnifyBox];
GraphicsDefs.ReleaseBox[@MagnifyBox];
GraphicsDefs.DestroyBox[@MagnifyBox,FALSE];
IF MagnifyList # NIL THEN SystemDefs.FreeSegment[MagnifyList];
MagnifyList _ NIL;
MagnifyListCount _ 0;
MagnifyListMax _ 0;
END;

SetMagnification  : PUBLIC PROCEDURE [bitsPerInch: CARDINAL] =
BEGIN
MagnifyBitsPerInch _ bitsPerInch;
END;

--display coord: micas to bits
dc: PROCEDURE [m: Mica] RETURNS [INTEGER] = INLINE
BEGIN
RETURN[PressDefs.MulDiv[m,bitsPerInch,micasPerInch]];
END;
--with rounding
dcr: PROCEDURE [m: Mica] RETURNS [INTEGER] = INLINE
BEGIN
RETURN[MulDivCeiling[m,bitsPerInch,micasPerInch]];
END;
--magnified display coord: micas to magnified bits
mdc: PROCEDURE [m: Mica] RETURNS [INTEGER] = INLINE
BEGIN
RETURN[PressDefs.SignedMulDiv[m,MagnifyBitsPerInch,micasPerInch]];
END;
--mica coord: bits to micas
mc: PROCEDURE [c: CARDINAL] RETURNS [Mica] = INLINE
BEGIN
RETURN[PressDefs.SignedMulDiv  [c,micasPerInch,bitsPerInch]];
END;
mcr: PROCEDURE [c: CARDINAL] RETURNS [Mica] = INLINE
BEGIN
RETURN[MulDivCeiling[c,micasPerInch,bitsPerInch]];
END;
--magnified mica coord: magnified bits to micas
mmc: PROCEDURE [c: CARDINAL] RETURNS [Mica] = INLINE
BEGIN
RETURN[PressDefs.MulDiv  [c,micasPerInch,MagnifyBitsPerInch]];
END;
--magnified: bits to magnified bits
m: PROCEDURE [c: INTEGER] RETURNS [INTEGER] = INLINE
BEGIN
RETURN [PressDefs.SignedMulDiv[c,MagnifyBitsPerInch,bitsPerInch]];
END;

SetMagnifierPosition  : PUBLIC PROCEDURE [x,y: Mica] =
BEGIN
MagnifyX _ dc[x];
MagnifyY _ dc[y];
END;

SetMagnifierSize  : PUBLIC PROCEDURE [width,height: CARDINAL] =
BEGIN
MagnifyWidth _ width;
```

```
MagnifyHeight _ height;
END;

MagToScreenCoords  : PUBLIC PROCEDURE[x,y: CARDINAL] RETURNS[rx,ry: CARDINAL] =
BEGIN
mbpi: CARDINAL = MagnifyBitsPerInch;
bpi: CARDINAL = bitsPerInch;
IF (NOT DisplayListDefs.Inside[x,y,@AdjustBox]) AND
                DisplayListDefs.Inside[x,y,@MagnifyBox] THEN
      RETURN[MagnifyX+PressDefs.SignedMulDiv[x-MagnifyX,bpi,mbpi],
                      MagnifyY+PressDefs.SignedMulDiv[y-MagnifyY,bpi,mbpi]];
RETURN[x,y];
END;

MagToMicaCoords  : PUBLIC PROCEDURE[x,y: CARDINAL] RETURNS[rx,ry: Mica] =
BEGIN OPEN PressDefs;
bpi: CARDINAL = bitsPerInch;
mbpi: CARDINAL = MagnifyBitsPerInch;
mpi: CARDINAL = micasPerInch;
IF (NOT DisplayListDefs.Inside[x,y,@AdjustBox]) AND
                DisplayListDefs.Inside[x,y,@MagnifyBox] THEN
      RETURN[MulDiv[MagnifyX,mpi,bpi]+SignedMulDiv[x-MagnifyX,mpi,mbpi],
                      MulDiv[MagnifyY,mpi,bpi]+SignedMulDiv[y-MagnifyY,mpi,mbpi]]
ELSE RETURN[MulDiv[x,mpi,bpi],MulDiv[y,mpi,bpi]];
END;


END.
```

```
DIRECTORY
    DisplayListDefs,
    GraphicsDefs,
    HalftoneDefs,
    IODefs,
    PressDefs,
    StrawDefs,
    StreamDefs,
    SystemDefs;

StrawImage : PROGRAM IMPORTS DisplayListDefs, GraphicsDefs, HalftoneDefs, IODefs, PressDefs, StrawDefs, SystemDefs
    EXPORTS StrawDefs =
BEGIN
Buttons: TYPE = MACHINE DEPENDENT RECORD
    [ KeyPadAndGarbage: [0..10000B),
      Mark: BOOLEAN, --TRUE means button up
      Select: BOOLEAN,
      Draw: BOOLEAN
    ];
KeyWord2: TYPE = MACHINE DEPENDENT RECORD
    [ One,Esc,Tab,F,Ctrl,C,J,B,Z,LShift,Period,Semi,Return,LArrow,Del,xxx: BOOLEAN
    ];
KeyWord: POINTER TO KeyWord2 = LOOPHOLE[177036B];
MouseButtons: POINTER TO Buttons = LOOPHOLE[177030B];
CursorX: POINTER TO CARDINAL = LOOPHOLE[426B];
CursorY: POINTER TO CARDINAL = LOOPHOLE[427B];
keys: StreamDefs.StreamHandle _ IODefs.GetInputStream[];
ScreenHeight: CARDINAL _ GraphicsDefs.GetDefaultBitmapHandle[].nLines;
selectGray: CARDINAL _ 15;

bitsPerInch: CARDINAL _ 72;
micasPerInch: CARDINAL = 2540;
Mica: TYPE = CARDINAL;
dc: PROCEDURE [m: Mica] RETURNS [CARDINAL] = INLINE
BEGIN
RETURN[PressDefs.SignedMulDiv[m,bitsPerInch,micasPerInch]];
END;

mc: PROCEDURE [c: CARDINAL] RETURNS [Mica] = INLINE
BEGIN
RETURN[PressDefs.MulDiv[c,micasPerInch,bitsPerInch]];
END;

EditGrayImage : PUBLIC PROCEDURE [do: POINTER TO StrawDefs.DotObject] =
BEGIN
pdd: POINTER TO PressDefs.PressDotsData _ do.DotsData;
borderWidth: CARDINAL _ MIN[16,
    MIN[do.Box.boxBitmap.nBits,do.Box.boxBitmap.nLines]/6];
x: INTEGER _ CursorX^;
y: INTEGER _ CursorY^;
RealCursor: GraphicsDefs.Bitmap _ [bank: 0,nWords: 1,nBits:16,nLines:16, bits:LOOPHOLE[431B]];
Cursor: GraphicsDefs.Bitmap _ [bank: 0,nWords: 1,nBits:16,nLines:16, bits:SystemDefs.AllocateHeapNode[16]];
xThird,yThird: CARDINAL;
WhiteBox: GraphicsDefs.Box;
GraphicsDefs.CreateBox[@WhiteBox,18,18];
GraphicsDefs.DisplayBox[@WhiteBox,x-1,y-1];

WHILE x IN [INTEGER[do.Box.displayX-borderWidth]..
                INTEGER[do.Box.displayX+do.Box.boxBitmap.nBits+borderWidth]] AND
    y IN [INTEGER[do.Box.displayY-borderWidth]..
            INTEGER[do.Box.displayY+do.Box.boxBitmap.nLines+borderWidth]] DO
    --check which edit border
    xThird _ MIN[2,(x-(do.Box.displayX-borderWidth))/
                    ((do.Box.boxBitmap.nBits+borderWidth+1)/3)];
    yThird _ MIN[2,(y-(do.Box.displayY-borderWidth))/
                    ((do.Box.boxBitmap.nLines+borderWidth+1)/3)];

    SELECT TRUE FROM
    Bounded[do,x,y,0,borderWidth] => --stretch border
        BEGIN
        MakeArrow[SELECT xThird FROM 0=>1,1=>7,ENDCASE=>14,
            SELECT yThird FROM 0=>1,1=>7,ENDCASE=>14,
```

```
            SELECT xThird FROM 0=>14,1=>7,ENDCASE=>1,
            SELECT yThird FROM 0=>14,1=>7,ENDCASE=>1,FALSE,@Cursor];
        GraphicsDefs.MoveBox[@WhiteBox,x-1,y-1];
        GraphicsDefs.ReplaceBitmap[@Cursor,0,0,@RealCursor];
        IF NOT MouseButtons.Select THEN
            BEGIN
            GraphicsDefs.RemoveBox[@WhiteBox];
            DisplayListDefs.DeleteDisplayItem[@do.Box];
            StrawDefs.RestoreGrid[@do.Box]; --must have correct bit counts
            XorBox[do];
            UNTIL MouseButtons.Select DO
                IF yThird = 1 OR xThird = 1 THEN StretchEdge[do,yThird*3+xThird]
                ELSE StretchCorner[do,yThird*3+xThird];
            ENDLOOP;
            XorBox[do];
            ReDisplay[do];
            END;
        END;
    ((xThird # 1) OR (yThird # 1)) AND
    Bounded[do,x,y,-borderWidth,borderWidth] => --window border
        BEGIN
        MakeArrow[SELECT xThird FROM 0=>0,1=>7,ENDCASE=>15,
            SELECT yThird FROM 0=>0,1=>7,ENDCASE=>15,
            SELECT xThird FROM 0=>8,1=>7,ENDCASE=>7,
            SELECT yThird FROM 0=>8,1=>7,ENDCASE=>7,TRUE,@Cursor];
        GraphicsDefs.MoveBox[@WhiteBox,x-1,y-1];
        GraphicsDefs.ReplaceBitmap[@Cursor,0,0,@RealCursor];
        IF NOT MouseButtons.Select THEN
            BEGIN
            newDo: StrawDefs.DotObject _ do^; --copy it
            dd: PressDefs.PressDotsData _ do.DotsData^;
            dx1,dx2,dy1,dy2: INTEGER;
            ndd: POINTER TO PressDefs.PressDotsData _ newDo.DotsData;
            nb: POINTER TO GraphicsDefs.Box _ @newDo.Box;
            nbm: POINTER TO GraphicsDefs.Bitmap _ @nb.boxBitmap;
            bMode: CARDINAL _ pdd.mode/4;
                        sMode: CARDINAL _ pdd.mode MOD 4;
                        normal: BOOLEAN = sMode IN [2..3];
            PtoD: PROCEDURE [pixels: INTEGER] RETURNS [dots: INTEGER] = INLINE
            BEGIN
            dots _ PressDefs.SignedMulDiv[pixels,
                dc[IF normal THEN dd.micaWidth ELSE dd.micaHeight],
                dd.displayPixels];
            END;
            DtoP: PROCEDURE [dots: INTEGER] RETURNS [pixels: INTEGER] = INLINE
            BEGIN
            pixels _ PressDefs.SignedMulDiv[dots,dd.displayPixels,
                dc[IF normal THEN dd.micaWidth ELSE dd.micaHeight]];
            END;
            LtoD: PROCEDURE [lines: INTEGER] RETURNS [dots: INTEGER] = INLINE
            BEGIN
            dots _ PressDefs.SignedMulDiv[lines,
                dc[IF normal THEN dd.micaHeight ELSE dd.micaWidth],
                dd.displayLines];
            END;
            DtoL: PROCEDURE [dots: INTEGER] RETURNS [lines: INTEGER] = INLINE
            BEGIN
            lines _ PressDefs.SignedMulDiv[dots,dd.displayLines,
                dc[IF normal THEN dd.micaHeight ELSE dd.micaWidth]];
            END;
            GraphicsDefs.RemoveBox[@WhiteBox];
            XorBox[do];
            UNTIL MouseButtons.Select DO
                WindowDots[@newDo,yThird*3+xThird];
            ENDLOOP;
            XorBox[@newDo];
            DisplayListDefs.DeleteDisplayItem[@do.Box];
            StrawDefs.RestoreGrid[@do.Box]; --must have correct bit counts
            --calculate new window
            dx1 _ nb.displayX-do.Box.displayX;
            dy1 _ nb.displayY-do.Box.displayY;
            dx2 _ (do.Box.displayX+do.Box.boxBitmap.nBits)-
```

```
                    (nb.displayX+nbm.nBits);
                dy2 _ (do.Box.displayY+do.Box.boxBitmap.nLines)-
                    (nb.displayY+nbm.nLines);
                ndd.passPixels _ MAX[0,INTEGER[dd.passPixels+DtoP[SELECT bMode FROM
                    0=>dx1, 1=>dx2, 2=>dy2, ENDCASE=>dy1]]];
                ndd.passLines _ MAX[0,INTEGER[dd.passLines+DtoL[SELECT sMode FROM
                    0=>dx1, 1=>dx2, 2=>dy2, ENDCASE=>dy1]]];
                ndd.displayPixels _ MIN[ndd.nPixels-ndd.passPixels,
                    dd.displayPixels+dd.passPixels-ndd.passPixels-
                    DtoP[SELECT bMode FROM 0=>dx2, 1=>dx1, 2=>dy1, ENDCASE=>dy2]];
                ndd.displayLines _ MIN[ndd.nLines-ndd.passLines,
                    dd.displayLines+dd.passLines-ndd.passLines-
                    DtoL[SELECT sMode FROM 0=>dx2, 1=>dx1, 2=>dy1, ENDCASE=>dy2]];
                IF normal THEN
                    BEGIN
                    nbm.nBits _ PtoD[ndd.displayPixels];
                    nbm.nLines _ LtoD[ndd.displayLines];
                    nb.displayX _ do.Box.displayX+
                        (IF bMode=0 THEN PtoD[ndd.passPixels-dd.passPixels]
                        ELSE PtoD[(dd.passPixels+dd.displayPixels)-
                            (ndd.passPixels+ndd.displayPixels)]);
                    nb.displayY _  do.Box.displayY+
                        (IF sMode=3 THEN LtoD[ndd.passLines-dd.passLines]
                        ELSE LtoD[(dd.passLines+dd.displayLines)-
                            (ndd.passLines+ndd.displayLines)]);
                    END
                ELSE
                    BEGIN
                    nbm.nBits _ LtoD[ndd.displayLines];
                    nbm.nLines _ PtoD[ndd.displayPixels];
                    nb.displayX _ do.Box.displayX+
                        (IF sMode=0 THEN LtoD[ndd.passLines-dd.passLines]
                        ELSE LtoD[(dd.passLines+dd.displayLines)-
                            (ndd.passLines+ndd.displayLines)]);
                    nb.displayY _  do.Box.displayY+
                        (IF bMode=3 THEN PtoD[ndd.passPixels-dd.passPixels]
                        ELSE PtoD[(dd.passPixels+dd.displayPixels)-
                            (ndd.passPixels+ndd.displayPixels)]);
                    END;
                do^ _ newDo;
                ReDisplay[do];
                END;
            END;
    --check rotation border
        Bounded[do,x,y,-borderWidth*2,borderWidth] =>
            BEGIN
            xLeft: BOOLEAN _
                CARDINAL[x]<do.Box.displayX+(do.Box.boxBitmap.nBits+borderWidth)/2;
            yTop: BOOLEAN _
                CARDINAL[y]<do.Box.displayY+(do.Box.boxBitmap.nLines+borderWidth)/2;
            quadrant: CARDINAL _ SELECT TRUE FROM
                xLeft AND yTop=>1,
                xLeft=>2,
                yTop=>0,
                ENDCASE=>3;
            MakeRotateCursor[quadrant,@Cursor];
            GraphicsDefs.MoveBox[@WhiteBox,x-1,y-1];
            GraphicsDefs.ReplaceBitmap[@Cursor,0,0,@RealCursor];
            IF NOT MouseButtons.Select THEN
                BEGIN
                bMode: [0..3] _ do.DotsData.mode/4;
                sMode: [0..3] _ do.DotsData.mode MOD 4;
                GraphicsDefs.RemoveBox[@WhiteBox];
                DisplayListDefs.DeleteDisplayItem[@do.Box];
                StrawDefs.RestoreGrid[@do.Box]; --must have correct bit counts
                IF quadrant = 3 THEN
                    BEGIN bMode _ Mirror[bMode];sMode _ Mirror[sMode];END
                ELSE
                    BEGIN
                    THROUGH [0..quadrant] DO
                        bMode _ R90[bMode];sMode _ R90[sMode];
                    ENDLOOP;
```

```
                    IF quadrant=0 OR quadrant=2 THEN --flip width, height
                        BEGIN
                        t: CARDINAL _ do.Box.boxBitmap.nBits;
                        do.Box.boxBitmap.nBits _ do.Box.boxBitmap.nLines;
                        do.Box.boxBitmap.nLines _ t;
                        END;
                    END;
                do.DotsData.mode _ bMode*4+sMode;
                ReDisplay[do];
                END;
            END;
        ENDCASE => --selection area
            BEGIN
            StrawDefs.NormalCursor[];
            GraphicsDefs.RemoveBox[@WhiteBox];
            IF NOT MouseButtons.Select THEN EXIT;
            END;

        --check for Draw button (gray scale adjust)
        IF (NOT MouseButtons.Draw) AND KeyWord.Ctrl AND KeyWord.LShift THEN
            BEGIN
            StrawDefs.NormalCursor[];
            GraphicsDefs.RemoveBox[@WhiteBox];
            SetGrayLevels[do];
            DisplayListDefs.DeleteDisplayItem[@do.Box];
            StrawDefs.RestoreGrid[@do.Box]; --must have correct bit counts
            ReDisplay[do];
            END;
        --check non-select button
        IF (NOT MouseButtons.Mark) OR (NOT MouseButtons.Draw) OR
            (NOT keys.endof[keys]) THEN EXIT;

        --update x,y
        x _ CursorX^;
        y _ CursorY^;
    ENDLOOP;
    GraphicsDefs.DestroyBox[@WhiteBox,TRUE];
    SystemDefs.FreeHeapNode[Cursor.bits];
    StrawDefs.NormalCursor[];
    END;

    Mirror : PROCEDURE [mode: [0..3]] RETURNS [[0..3]] = INLINE
    BEGIN
    RETURN[SELECT mode FROM 0=>1,1=>0,2=>2,ENDCASE=>3];
    END;

    R90 : PROCEDURE [mode: [0..3]] RETURNS [[0..3]] = INLINE
    BEGIN
    RETURN[SELECT mode FROM 0=>2,1=>3,2=>1,ENDCASE=>0];
    END;

    ReDisplay : PROCEDURE [do: POINTER TO StrawDefs.DotObject] =
    BEGIN
    StrawDefs.WaitCursor[];
    GraphicsDefs.DestroyBox[@do.Box,FALSE];
    do.DotsData.micaWidth _ mc[do.Box.boxBitmap.nBits+1];
    do.DotsData.micaHeight _ mc[do.Box.boxBitmap.nLines+1];
    do.Object.micaX _ mc[do.Box.displayX];
    do.Object.micaY _
        mc[ScreenHeight-(do.Box.displayY+do.Box.boxBitmap.nLines+1)];
    StrawDefs.CreateDotsBox[do];
    DisplayListDefs.AddDisplayItem[@do.Box];
    GraphicsDefs.SetGrayLevel[selectGray];
    GraphicsDefs.XorGray[0,0,do.Box.boxBitmap.nBits,do.Box.boxBitmap.nLines,
        @do.Box.boxBitmap];
    GraphicsDefs.DisplayBox[@do.Box,do.Box.displayX,do.Box.displayY];
    StrawDefs.NormalCursor[];
    END;

    XorBox : PROCEDURE [do: POINTER TO StrawDefs.DotObject] =
    BEGIN
    xl: CARDINAL _ do.Box.displayX;
```

```
xr: CARDINAL _ xl+do.Box.boxBitmap.nBits;
yt: CARDINAL _ do.Box.displayY;
yb: CARDINAL _ yt+do.Box.boxBitmap.nLines;
GraphicsDefs.XorArea[xl,yt,xr,yt];
GraphicsDefs.XorArea[xr,yt,xr,yb];
GraphicsDefs.XorArea[xr,yb,xl,yb];
GraphicsDefs.XorArea[xl,yb,xl,yt];
END;

WindowDots  : PUBLIC PROCEDURE [do: POINTER TO StrawDefs.DotObject,
      octant: CARDINAL] =
BEGIN
b: POINTER TO GraphicsDefs.Box _ @do.Box;
bm: POINTER TO GraphicsDefs.Bitmap _ @b.boxBitmap;
dx: INTEGER _ b.displayX-CursorX^;
dy: INTEGER _ b.displayY-CursorY^;
XorBox[do];
SELECT octant MOD 3 FROM
0=> --left edge
IF INTEGER[bm.nBits+dx] > 0 THEN
      BEGIN bm.nBits _ bm.nBits+dx;b.displayX _ b.displayX-dx;END;
2=> --right edge
      bm.nBits _ MAX[1,-(dx+1)];
ENDCASE;
SELECT octant/3 FROM
0=> --top edge
IF INTEGER[bm.nLines+dy] > 0 THEN
      BEGIN bm.nLines _ bm.nLines+dy;b.displayY _ b.displayY-dy;END;
2=> --bottom edge
      bm.nLines _ MAX[1,-(dy+1)];
ENDCASE;
XorBox[do];
END;

StretchEdge  : PUBLIC PROCEDURE [do: POINTER TO StrawDefs.DotObject,
      octant: CARDINAL] =
BEGIN
b: POINTER TO GraphicsDefs.Box _ @do.Box;
bm: POINTER TO GraphicsDefs.Bitmap _ @b.boxBitmap;
dx: INTEGER _ b.displayX-CursorX^;
dy: INTEGER _ b.displayY-CursorY^;
XorBox[do];
SELECT octant FROM
1=> --top edge
IF INTEGER[bm.nLines+dy] > 0 THEN
      BEGIN bm.nLines _ bm.nLines+dy;b.displayY _ b.displayY-dy;END;
3=> --left edge
IF INTEGER[bm.nBits+dx] > 0 THEN
      BEGIN bm.nBits _ bm.nBits+dx;b.displayX _ b.displayX-dx;END;
5=> --right edge
      bm.nBits _ MAX[1,-(dx+1)];
7=> --bottom edge
      bm.nLines _ MAX[1,-(dy+1)];
ENDCASE=>ERROR;
XorBox[do];
END;

StretchCorner  : PUBLIC PROCEDURE [do: POINTER TO StrawDefs.DotObject,
      octant: CARDINAL] =
BEGIN
b: POINTER TO GraphicsDefs.Box _ @do.Box;
bm: POINTER TO GraphicsDefs.Bitmap _ @b.boxBitmap;
xRatio: CARDINAL _ do.DotsData.micaWidth;
yRatio: CARDINAL _ do.DotsData.micaHeight;
calculatedX,calculatedY: INTEGER;
dx,dy: INTEGER;

XorBox[do];
dx _ MAX[1,INTEGER[SELECT octant MOD 3 FROM
      0=>b.displayX+bm.nBits+1-CursorX^,2=>CursorX^-b.displayX,ENDCASE=>ERROR]];
dy _ MAX[1,INTEGER[SELECT octant/3 FROM
      0=>b.displayY+bm.nLines+1-CursorY^,2=>CursorY^-b.displayY,ENDCASE=>ERROR]];
```

```
calculatedX _ PressDefs.MulDiv[dy,xRatio,yRatio];
calculatedY _ PressDefs.MulDiv[dx,yRatio,xRatio];
IF ABS[calculatedX-dx] > ABS[calculatedY-dy]
      THEN dy _ calculatedY ELSE dx _ calculatedX;
IF octant MOD 3 = 0 THEN b.displayX _ b.displayX-(dx-bm.nBits);
IF octant/3 = 0 THEN b.displayY _ b.displayY-(dy-bm.nLines);
bm.nBits _ dx;
bm.nLines _ dy;
XorBox[do];
END;

MakeRotateCursor   : PROCEDURE [quadrant: CARDINAL,cursor: POINTER TO GraphicsDefs.Bitmap] =
BEGIN
IF quadrant=3 THEN --mirror icon
      BEGIN
      MakeArrow[2,7,13,7,FALSE,cursor];
      GraphicsDefs.PutArea[7,1,8,14,cursor];
      RETURN;
      END;
GraphicsDefs.EraseArea[0,0,16,16,cursor];
GraphicsDefs.PutArea[4,4,11,5,cursor];
IF quadrant>0 THEN GraphicsDefs.PutArea[4,4,5,11,cursor];
IF quadrant>1 THEN GraphicsDefs.PutArea[4,10,11,11,cursor];
SELECT quadrant FROM
      0=>
            BEGIN
            GraphicsDefs.PutArea[5,3,5,6,cursor];
            GraphicsDefs.PutArea[6,2,6,7,cursor];
            END;
      1=>
            BEGIN
            GraphicsDefs.PutArea[2,9,7,9,cursor];
            GraphicsDefs.PutArea[3,10,6,10,cursor];
            END;
      ENDCASE=>
            BEGIN
            GraphicsDefs.PutArea[9,8,9,13,cursor];
            GraphicsDefs.PutArea[10,9,10,12,cursor];
            END;
END;

MakeArrow  : PROCEDURE [x1,y1,x2,y2: CARDINAL,box: BOOLEAN,cursor: POINTER TO GraphicsDefs.Bitmap] =
BEGIN
GraphicsDefs.EraseArea[0,0,16,16,cursor];
SELECT TRUE FROM
      x1 = x2 => --vertical
            BEGIN
            t: CARDINAL;
            IF y1 > y2 THEN BEGIN t _ y1;y1 _ y2;y2 _ t;END;
            GraphicsDefs.PutArea[x1,y1,x1+1,y2,cursor];
            GraphicsDefs.PutArea[x1-1,y1+1,x1+2,y1+1,cursor];
            GraphicsDefs.PutArea[x1-2,y1+2,x1+3,y1+2,cursor];
            GraphicsDefs.PutArea[x1-1,y2-1,x1+2,y2-1,cursor];
            GraphicsDefs.PutArea[x1-2,y2-2,x1+3,y2-2,cursor];
            END;
      y1 = y2 => --horizontal
            BEGIN
            t: CARDINAL;
            IF x1 > x2 THEN BEGIN t _ x1;x1 _ x2;x2 _ t;END;
            GraphicsDefs.PutArea[x1,y1,x2,y1+1,cursor];
            GraphicsDefs.PutArea[x1+1,y1-1,x1+1,y1+2,cursor];
            GraphicsDefs.PutArea[x1+2,y1-2,x1+2,y1+3,cursor];
            GraphicsDefs.PutArea[x2-1,y1-1,x2-1,y1+2,cursor];
            GraphicsDefs.PutArea[x2-2,y1-2,x2-2,y1+3,cursor];
            END;
      ENDCASE => --diagonal
            BEGIN
            dx: INTEGER _ IF x1<x2 THEN 1 ELSE -1;
            dy: INTEGER _ IF y1<y2 THEN 1 ELSE -1;
            GraphicsDefs.PutLine[x1,y1,x2,y2,cursor];
            GraphicsDefs.PutLine[x1+dx,y1,x2,y2-dy,cursor];
            GraphicsDefs.PutLine[x1,y1+dy,x2-dx,y2,cursor];
```

```
            GraphicsDefs.PutLine[x1,y1,x1+4*dx,y1,cursor];
            GraphicsDefs.PutLine[x2-4*dx,y2,x2,y2,cursor];
            GraphicsDefs.PutLine[x1,y1,x1,y1+4*dy,cursor];
            GraphicsDefs.PutLine[x2,y2-4*dy,x2,y2,cursor];
            END;
IF box THEN
      BEGIN
      GraphicsDefs.PutLine[4,4,4,11,cursor];
      GraphicsDefs.PutLine[4,11,11,11,cursor];
      GraphicsDefs.PutLine[11,11,11,4,cursor];
      GraphicsDefs.PutLine[11,4,4,4,cursor];
      END;
END;

Bounded : PROCEDURE [do: POINTER TO StrawDefs.DotObject,x,y: INTEGER,
      lowBound,border: INTEGER] RETURNS [BOOLEAN] =
BEGIN
highBound: INTEGER _ lowBound+border;
lx: INTEGER _ do.Box.displayX-highBound;
ty: INTEGER _ do.Box.displayY-highBound;
rx: INTEGER _ do.Box.displayX+do.Box.boxBitmap.nBits-1+highBound-border;
by: INTEGER _ do.Box.displayY+do.Box.boxBitmap.nLines-1+highBound-border;

IF x IN [lx..lx+border] AND y IN [ty..by] THEN RETURN[TRUE]; --left edge
IF x IN [rx-border..rx] AND y IN [ty..by] THEN RETURN[TRUE]; --right edge
IF x IN [lx..rx] AND y IN [ty..ty+border] THEN RETURN[TRUE]; --top edge
IF x IN [lx..rx] AND y IN [by-border..by] THEN RETURN[TRUE]; --bottom edge

RETURN[FALSE];
END;

--adjust gray scale range routines
SetGrayLevels  : PROCEDURE [do: POINTER TO StrawDefs.DotObject] =
BEGIN
x: CARDINAL;
AdjustBox: GraphicsDefs.Box;
AdjustBitmap: POINTER TO GraphicsDefs.Bitmap _ @AdjustBox.boxBitmap;
minSep: INTEGER _ 2;
xOffset: INTEGER _ 8;
eraseBoolean: BOOLEAN = TRUE;
black,white,blackX,whiteX: INTEGER;
blackTop,whiteTop: BOOLEAN;
source: PACKED ARRAY [0..256) OF [0..377B];
MakeRect  : PROCEDURE [x1,y1,x2,y2: CARDINAL,b: POINTER TO GraphicsDefs.Bitmap] =
BEGIN
GraphicsDefs.ReplaceArea[x1,y1,x1,y2,AdjustBitmap];
GraphicsDefs.ReplaceArea[x1,y2,x2,y2,AdjustBitmap];
GraphicsDefs.ReplaceArea[x2,y2,x2,y1,AdjustBitmap];
GraphicsDefs.ReplaceArea[x2,y1,x1,y1,AdjustBitmap];
END;

XorMarks  : PROCEDURE =
BEGIN
blackY: CARDINAL _ IF blackTop THEN 16 ELSE 32;
whiteY: CARDINAL _ IF whiteTop THEN 16 ELSE 32;
IF blackTop THEN
      GraphicsDefs.XorArea[blackX-7,blackY+8,blackX-1,blackY+15,AdjustBitmap]
ELSE
      GraphicsDefs.XorArea[blackX-7,blackY,blackX-1,blackY+7,AdjustBitmap];

GraphicsDefs.XorArea[blackX,blackY,blackX,blackY+15,AdjustBitmap];

IF whiteTop THEN
      BEGIN
      GraphicsDefs.XorArea[whiteX+1,whiteY+8,whiteX+7,whiteY+15,AdjustBitmap];
      GraphicsDefs.XorArea[whiteX+1,whiteY+9,whiteX+6,whiteY+14,AdjustBitmap];
      END
ELSE
      BEGIN
      GraphicsDefs.XorArea[whiteX+1,whiteY,whiteX+7,whiteY+7,AdjustBitmap];
      GraphicsDefs.XorArea[whiteX+1,whiteY+1,whiteX+6,whiteY+6,AdjustBitmap];
      END;
```

```
GraphicsDefs.XorArea[whiteX,whiteY,whiteX,whiteY+15,AdjustBitmap];
END;

SetBlackWhite  : PROCEDURE =
BEGIN
SELECT TRUE FROM
      blackTop AND whiteTop =>
            BEGIN
            white _ 255+PressDefs.MulDiv[255-(whiteX-xOffset),255,whiteX-blackX];
            black _ -PressDefs.MulDiv[blackX-xOffset,255,whiteX-blackX];
            END;
      blackTop =>
            BEGIN
            white _ whiteX - xOffset;
            black _ white-PressDefs.MulDiv[255,white,255-(blackX-xOffset)];
            IF black > 255 THEN black _ -32000; --overflow on MulDiv
            END;
      whiteTop =>
            BEGIN
            black _ blackX - xOffset;
            white _ black+PressDefs.MulDiv[255-black,255,(whiteX-xOffset)];
            IF white < 0 THEN white _ 32000; --overflow on MulDiv
            END;
      ENDCASE =>
            BEGIN
            black _ blackX - xOffset;
            white _ whiteX - xOffset;
            END;
IF white <= black THEN ERROR;
GraphicsDefs.EraseArea[xOffset,48+1,xOffset+255,48+14,AdjustBitmap];
HalftoneDefs.InitHalftone[xOffset,48+1,256,256,black,white,AdjustBitmap,1,14];
[] _ HalftoneDefs.PrintHalftoneLine[@source];
XorMarks[];
GraphicsDefs.MoveBox[@AdjustBox,AdjustBox.displayX,AdjustBox.displayY];
END;

BlackAdjust  : PROCEDURE =
BEGIN
x: INTEGER _ CursorX^-AdjustBox.displayX;
XorMarks[];
blackX _ MIN[254,MAX[0,x-xOffset]]+xOffset;
SELECT (CursorY^-AdjustBox.displayY)/16 FROM
      1=> --top adjust
            BEGIN
            IF whiteTop THEN
                  IF whiteX-minSep < blackX THEN
                        BEGIN
                        blackX _ MIN[(255-minSep)+xOffset,blackX];
                        whiteX _ blackX+minSep;
                        END;
            blackTop _ TRUE;
            SetBlackWhite[];
            END;
      2=> --bottom adjust
            BEGIN
            IF NOT whiteTop THEN
                  IF whiteX <= blackX THEN whiteX _ blackX+1;
            blackTop _ FALSE;
            SetBlackWhite[];
            END;
      ENDCASE => XorMarks[];
END;

WhiteAdjust  : PROCEDURE =
BEGIN
x: INTEGER _ CursorX^-AdjustBox.displayX;
XorMarks[];
whiteX _ MIN[255,MAX[1,x-xOffset]]+xOffset;
SELECT (CursorY^-AdjustBox.displayY)/16 FROM
      1=> --top adjust
            BEGIN
```

```
            IF blackTop THEN
                IF blackX+minSep > whiteX THEN
                    BEGIN
                    whiteX _ MAX[minSep+xOffset,whiteX];
                    blackX _ whiteX-minSep;
                    END;
            whiteTop _ TRUE;
            SetBlackWhite[];
            END;
        2=> --bottom adjust
            BEGIN
            IF NOT blackTop THEN
                IF blackX >= whiteX THEN blackX _ whiteX-1;
            whiteTop _ FALSE;
            SetBlackWhite[];
            END;
        ENDCASE => XorMarks[];
END;

SetBlackXWhiteX   : PROCEDURE =
BEGIN
SELECT TRUE FROM
        blackTop AND whiteTop =>
            BEGIN
            blackX _ 255 - PressDefs.MulDiv[white,255,white-black] + xOffset;
            whiteX _ PressDefs.MulDiv[255-black,255,white-black] + xOffset;
            END;
        blackTop =>
            BEGIN
            whiteX _ white + xOffset;
            blackX _ 255 - PressDefs.MulDiv[white,255,white-black] + xOffset;
            END;
        whiteTop =>
            BEGIN
            blackX _ black + xOffset;
            whiteX _ PressDefs.MulDiv[255-black,255,white-black] + xOffset;
            END;
        ENDCASE =>
            BEGIN
            blackX _ black + xOffset;
            whiteX _ white + xOffset;
            END;
END;

--set up global variables
black _ do.DotsData.min;
white _ do.DotsData.max;
blackTop _ black < 0;
whiteTop _ white > 255;
SetBlackXWhiteX[];

--talk to user
FOR x IN [0..256) DO source[x] _ x;ENDLOOP;
GraphicsDefs.CreateBox[@AdjustBox,256+16,16*6];
MakeRect[0,0,256+16-1,16*6-1,AdjustBitmap];
MakeRect[xOffset-1,0,xOffset+256,15,AdjustBitmap];
MakeRect[xOffset-1,48,xOffset+256,48+15,AdjustBitmap];
HalftoneDefs.InitHalftone[xOffset,1,256,256,0,255,AdjustBitmap,1,14];
[] _ HalftoneDefs.PrintHalftoneLine[@source];
HalftoneDefs.InitHalftone[xOffset,48+1,256,256,black,white,AdjustBitmap,1,14];
[] _ HalftoneDefs.PrintHalftoneLine[@source];
XorMarks[];

GraphicsDefs.DisplayBox[@AdjustBox,CursorX^-272/2,CursorY^-16*6/2];
WHILE GraphicsDefs.CursorInBox[@AdjustBox] DO
        BEGIN
        x: INTEGER _ CursorX^-AdjustBox.displayX;
        IF MouseButtons.Mark THEN LOOP;
        SELECT (CursorY^-AdjustBox.displayY)/16 FROM
            0=> LOOP; --full range display
            1=> --top adjust
                BEGIN
```

```
                IF blackTop THEN
                    BEGIN
                    IF x IN [blackX-8..blackX] THEN
                        BEGIN
                        WHILE NOT MouseButtons.Mark DO BlackAdjust[];ENDLOOP;
                        LOOP;
                        END;
                    END;
                IF whiteTop THEN
                    BEGIN
                    IF x IN [whiteX..whiteX+8] THEN
                        BEGIN
                        WHILE NOT MouseButtons.Mark DO WhiteAdjust[];ENDLOOP;
                        LOOP;
                        END;
                    END;
                END;
            2=> --bottom adjust
                BEGIN
                IF NOT blackTop THEN
                    BEGIN
                    IF x IN [blackX-8..blackX] THEN
                        BEGIN
                        WHILE NOT MouseButtons.Mark DO BlackAdjust[];ENDLOOP;
                        LOOP;
                        END;
                    END;
                IF NOT whiteTop THEN
                    BEGIN
                    IF x IN [whiteX..whiteX+8] THEN
                        BEGIN
                        WHILE NOT MouseButtons.Mark DO WhiteAdjust[];ENDLOOP;
                        LOOP;
                        END;
                    END;
                END;
            ENDCASE=> LOOP; --nothing
        END;
    ENDLOOP;

    do.DotsData.min _ black;
    do.DotsData.max _ white;
    GraphicsDefs.DestroyBox[@AdjustBox,TRUE];
    END;

    END.
```

```
DIRECTORY
    GraphicsDefs: FROM "GraphicsDefs";

DisplayListDefs: DEFINITIONS =
BEGIN

GetDisplayListHandle   : PROCEDURE RETURNS [POINTER TO ARRAY [0..0) OF POINTER TO GraphicsDefs.Box];
GetDisplayCount  : PUBLIC PROCEDURE RETURNS [CARDINAL];
ResetDisplayCount   : PUBLIC PROCEDURE [nItems: CARDINAL _ 0];
AddDisplayItem   : PROCEDURE[B: POINTER TO GraphicsDefs.Box];
DeleteDisplayItem:    PROCEDURE [B: POINTER TO GraphicsDefs.Box];
ReDisplay  : PROCEDURE [B: POINTER TO GraphicsDefs.Box];
Intersection   : PROCEDURE[b1,b2: POINTER TO GraphicsDefs.Box] RETURNS [BOOLEAN];
Inside : PROCEDURE[x,y: CARDINAL,b: POINTER TO GraphicsDefs.Box] RETURNS [BOOLEAN];

END.
```

```
DIRECTORY
    DisplayListDefs: FROM "DisplayListDefs",
    GraphicsDefs: FROM "GraphicsDefs",
    InlineDefs: FROM "InlineDefs" USING [COPY],
    SystemDefs: FROM "SystemDefs" USING [AllocateSegment,FreeSegment];

DisplayList : PROGRAM IMPORTS GraphicsDefs, InlineDefs, SystemDefs EXPORTS DisplayListDefs =
BEGIN

displayCount: CARDINAL _ 0;
displayListMax: CARDINAL _ 0;
displayListIncrement: CARDINAL = 1024;
displayList: POINTER TO ARRAY [0..0) OF POINTER TO GraphicsDefs.Box _ NIL;

MakeLongPointer : PROCEDURE [ptr: POINTER,bank: CARDINAL] RETURNS [LONG POINTER] = MACHINE CODE BEGIN END;

GetDisplayListHandle : PUBLIC PROCEDURE RETURNS [POINTER TO ARRAY [0..0) OF POINTER TO GraphicsDefs.Box] =
BEGIN
RETURN[displayList];
END;

ResetDisplayCount : PUBLIC PROCEDURE [nItems: CARDINAL _ 0] =
BEGIN
displayCount _ nItems;
END;

GetDisplayCount : PUBLIC PROCEDURE RETURNS [CARDINAL] =
BEGIN
RETURN[displayCount];
END;

AddDisplayItem : PUBLIC PROCEDURE[B: POINTER TO GraphicsDefs.Box] =
BEGIN
IF displayCount >= displayListMax THEN
    BEGIN
    newDisplayList: POINTER TO ARRAY [0..0) OF POINTER TO GraphicsDefs.Box;
    newDisplayList _ SystemDefs.AllocateSegment[displayListMax+displayListIncrement];
    InlineDefs.COPY[from: displayList,
        nwords: displayListMax+displayListIncrement,
        to: newDisplayList];
    IF displayList # NIL THEN SystemDefs.FreeSegment[displayList];
    displayList _ newDisplayList;
    displayListMax _ displayListMax+displayListIncrement;
    END;
displayList[displayCount] _ B;
displayCount _ displayCount+1;
END;

DeleteDisplayItem:  PUBLIC PROCEDURE [B: POINTER TO GraphicsDefs.Box] =
BEGIN
i,j: CARDINAL;
FOR i IN [0..displayCount) DO
    IF displayList[i] = B THEN
        BEGIN
        FOR j IN [i+1..displayCount) DO
            displayList[j-1] _ displayList[j];
        ENDLOOP;
        displayCount _ displayCount-1;
        IF B.savedBits # NIL THEN GraphicsDefs.RemoveBox[B]
        ELSE
            BEGIN
            ReDisplay[B];
            B.displayed _ FALSE;
            END;
        EXIT;
        END;
ENDLOOP;
END;

ReDisplay : PUBLIC PROCEDURE [B: POINTER TO GraphicsDefs.Box] =
BEGIN
i: CARDINAL;
```

```
GraphicsDefs.ReleaseBox[B];

--IF B.displayed THEN
    BEGIN
    Screen: POINTER TO GraphicsDefs.Bitmap _
        GraphicsDefs.GetDefaultBitmapHandle[];
    ScreenRect: GraphicsDefs.Rectangle _
        [MAX[INTEGER[B.displayX],0],MAX[INTEGER[B.displayY],0],
         MIN[B.displayX + B.boxBitmap.nBits - 1,Screen.nBits-1],
         MIN[B.displayY + B.boxBitmap.nLines - 1,Screen.nLines-1]
        ];
    GraphicsDefs.EraseArea[ScreenRect.x1,ScreenRect.y1,ScreenRect.x2,ScreenRect.y2];
    --and regenerate everyone else
    FOR i IN [0..displayCount) DO
        IF displayList[i].displayed AND Intersection[B,displayList[i]] THEN
            BEGIN
            C: POINTER TO GraphicsDefs.Box _ displayList[i];
            srcRect: GraphicsDefs.Rectangle;
            destRect: GraphicsDefs.Rectangle;

            destRect.x1 _ MAX[INTEGER[C.displayX],ScreenRect.x1];
            destRect.y1 _ MAX[INTEGER[C.displayY],ScreenRect.y1];
            destRect.x2 _ MIN[C.displayX + C.boxBitmap.nBits - 1,ScreenRect.x2];
            destRect.y2 _ MIN[C.displayY + C.boxBitmap.nLines - 1,ScreenRect.y2];

            srcRect.x1 _ destRect.x1 - C.displayX;
            srcRect.y1 _ destRect.y1 - C.displayY;
            srcRect.x2 _ srcRect.x1 + (destRect.x2 - destRect.x1);
            srcRect.y2 _ srcRect.y1 + (destRect.y2 - destRect.y1);
            IF C.type = gray THEN
                BEGIN
                GraphicsDefs.TransferRectangle[@C.boxBitmap,Screen,
                                        @srcRect,@destRect,erase];
                GraphicsDefs.SetGrayLevel[C.gray];
                GraphicsDefs.TransferRectangle[@C.boxBitmap,Screen,
                                        @srcRect,@destRect,paint,andgray];
                END
            ELSE GraphicsDefs.TransferRectangle[@C.boxBitmap,Screen,
                                        @srcRect,@destRect,C.function];
            END;
    ENDLOOP;
    END;
END;

Intersection  : PUBLIC PROCEDURE[b1,b2: POINTER TO GraphicsDefs.Box] RETURNS [BOOLEAN] =
BEGIN
x1: CARDINAL _ MAX[0,INTEGER[b1.displayX]];
x2: CARDINAL _ MAX[0,INTEGER[b2.displayX]];
x1a: CARDINAL _ b1.displayX + b1.boxBitmap.nBits - 1;
x2a: CARDINAL _ b2.displayX + b2.boxBitmap.nBits - 1;
y1: CARDINAL _ MAX[0,INTEGER[b1.displayY]];
y2: CARDINAL _ MAX[0,INTEGER[b2.displayY]];
y1a: CARDINAL _ b1.displayY + b1.boxBitmap.nLines - 1;
y2a: CARDINAL _ b2.displayY + b2.boxBitmap.nLines - 1;

IF x1a  < x2  OR y1a  < y2  THEN RETURN [FALSE];
IF x2a  < x1  OR y2a  < y1  THEN RETURN [FALSE];
RETURN [TRUE];
END;

Inside : PUBLIC PROCEDURE[x,y: CARDINAL,b: POINTER TO GraphicsDefs.Box] RETURNS [BOOLEAN] =
BEGIN
x1: CARDINAL _ MAX[0,INTEGER[b.displayX]];
x1a: CARDINAL _ b.displayX + b.boxBitmap.nBits - 1;
y1: CARDINAL _ MAX[0,INTEGER[b.displayY]];
y1a: CARDINAL _ b.displayY + b.boxBitmap.nLines - 1;

RETURN [ x IN [x1..x1a] AND y IN [y1..y1a] ];
END;

END.
```

```
-- magicDefs.Mesa

DIRECTORY
  AltoFileDefs: FROM "AltoFileDefs",
  GraphicsDefs: FROM "GraphicsDefs",
  Mopcodes: FROM "Mopcodes";

MagicDefs: DEFINITIONS =
  BEGIN

MakeLongPointer  : PUBLIC PROCEDURE [ptr: POINTER,bank: CARDINAL] RETURNS [LONG POINTER] = MACHINE CODE BEGIN
END;

Random : PROCEDURE RETURNS [CARDINAL];
GetUserBox  : PROCEDURE [xRatio,yRatio: CARDINAL] RETURNS [x,y: CARDINAL,dx,dy: INTEGER];
Menu : PROCEDURE [nItems: CARDINAL,itemList: POINTER TO ARRAY [0..0) OF STRING,font: POINTER TO
GraphicsDefs.StrikeFont] RETURNS [index: CARDINAL];
InitReadBlock  : PROCEDURE;
ReadBlock  : PROCEDURE[sfd: POINTER,p: POINTER TO UNSPECIFIED,nwords: CARDINAL] RETURNS [POINTER TO
UNSPECIFIED];

END.
```

```
DIRECTORY
      AltoFileDefs: FROM "AltoFileDefs",
      DiscoDefs: FROM "DiscoDefs" USING [SFD,ScanBuffer],
      GraphicsDefs: FROM "GraphicsDefs",
      IODefs: FROM "IODefs",
      InlineDefs: FROM "InlineDefs",
      MagicDefs: FROM "MagicDefs",
      Mopcodes: FROM "MopCodes";

Magic : PROGRAM IMPORTS DiscoDefs, InlineDefs, GraphicsDefs EXPORTS MagicDefs =
BEGIN

RandomTable: ARRAY [ 0..16) OF CARDINAL _
[1,22222,30303,7345,5525,6666,7777,877,9999,1770,11111,12,1313,7714,15151,1776];
next: CARDINAL _ 1;
Random : PUBLIC PROCEDURE RETURNS [CARDINAL] =
BEGIN
this: CARDINAL _ next;
next _ (next+1) MOD 16;
RandomTable[this] _ RandomTable[this]+RandomTable[next];
RETURN[RandomTable[this]];
END;
Buttons: TYPE = MACHINE DEPENDENT RECORD
      [ KeyPadAndGarbage: [0..10000B),
        Red: [0..1],
        Blue: [0..1],
        Yellow: [0..1]
      ];
MouseButtons: POINTER TO Buttons = LOOPHOLE[177030B];
CursorX: POINTER TO CARDINAL = LOOPHOLE[426B];
CursorY: POINTER TO CARDINAL = LOOPHOLE[427B];
GetUserBox : PUBLIC PROCEDURE [xRatio,yRatio: CARDINAL] RETURNS [x,y: CARDINAL,dx,dy: INTEGER] =
BEGIN
calculatedX,calculatedY: LONG INTEGER;
UNTIL MouseButtons.Red = 0 DO ENDLOOP;
x _ CursorX^;
y _ CursorY^;
dx _ dy _ 0;
DO
WHILE MouseButtons.Red = 0 DO
      GraphicsDefs.XorArea[x,y,x+dx,y];
      GraphicsDefs.XorArea[x+dx,y,x+dx,y+dy];
      GraphicsDefs.XorArea[x+dx,y+dy,x,y+dy];
      GraphicsDefs.XorArea[x,y+dy,x,y];
      dx _ CursorX^ - x;
      dy _ CursorY^ - y;
      calculatedX _ (LONG[dy-1]*LONG[xRatio])/LONG[yRatio];
      calculatedY _ (LONG[dx+1]*LONG[yRatio])/LONG[xRatio];
      IF ABS[ABS[calculatedX]-ABS[dx]] > ABS[ABS[calculatedY]-ABS[dy]]
            THEN dy _ (IF dy<0 THEN -1 ELSE 1)*InlineDefs.LowHalf[ABS[calculatedY]]
            ELSE dx _ (IF dx<0 THEN -1 ELSE 1)*InlineDefs.LowHalf[ABS[calculatedX]];
      GraphicsDefs.XorArea[x,y,x+dx,y];
      GraphicsDefs.XorArea[x+dx,y,x+dx,y+dy];
      GraphicsDefs.XorArea[x+dx,y+dy,x,y+dy];
      GraphicsDefs.XorArea[x,y+dy,x,y];
ENDLOOP;
IF dx < 0 THEN BEGIN x _ x+dx;dx _ ABS[dx];END;
IF dy < 0 THEN BEGIN y _ y+dy;dy _ ABS[dy];END;
WHILE MouseButtons.Yellow = 0 DO --move origin
      GraphicsDefs.XorArea[x,y,x+dx,y];
      GraphicsDefs.XorArea[x+dx,y,x+dx,y+dy];
      GraphicsDefs.XorArea[x+dx,y+dy,x,y+dy];
      GraphicsDefs.XorArea[x,y+dy,x,y];
      x _ CursorX^;
      y _ CursorY^;
      GraphicsDefs.XorArea[x,y,x+dx,y];
      GraphicsDefs.XorArea[x+dx,y,x+dx,y+dy];
      GraphicsDefs.XorArea[x+dx,y+dy,x,y+dy];
      GraphicsDefs.XorArea[x,y+dy,x,y];
ENDLOOP;
IF MouseButtons.Blue = 0 THEN RETURN[x,y,dx,dy];
```

```
ENDLOOP;
END;

Menu : PUBLIC PROCEDURE [nItems: CARDINAL,itemList: POINTER TO ARRAY [0..0) OF STRING,font: POINTER TO
GraphicsDefs.StrikeFont] RETURNS [index: CARDINAL] =
BEGIN
stringLen: PUBLIC PROCEDURE [s: STRING] RETURNS [CARDINAL] =
      BEGIN
      i: CARDINAL;
      widthList: POINTER TO ARRAY [0..0) OF CARDINAL _ font.xInSegment;
      len: CARDINAL _ 0;
      FOR i IN [0..s.length) DO
            BEGIN ix: CARDINAL _ LOOPHOLE[s[i],CARDINAL];
            len _ len + widthList[ix+1] - widthList[ix];
            END;
      ENDLOOP;
      RETURN[len];
      END;
menuWidth: CARDINAL _ 0;
fontHeight: CARDINAL _ font.ascent + font.descent;
menuHeight: CARDINAL _ nItems * fontHeight;
menuBox: GraphicsDefs.Box;
b: POINTER TO GraphicsDefs.Box _ @menuBox;
menuBitmap: POINTER TO GraphicsDefs.Bitmap _ @menuBox.boxBitmap;
Select: PUBLIC PROCEDURE [index: CARDINAL] =
      BEGIN
      GraphicsDefs.XorArea[0,index*fontHeight,
                  stringLen[itemList[index]],(index+1)*fontHeight,menuBitmap];
      GraphicsDefs.PaintBox[@menuBox];
      END;
i: CARDINAL;
Screen: POINTER TO GraphicsDefs.Bitmap _ GraphicsDefs.GetDefaultBitmapHandle[];
DCB: TYPE = MACHINE DEPENDENT RECORD
 [ link: POINTER TO DCB,
      lowResolution: BOOLEAN,
      inverseVideo: BOOLEAN,
      offset: [0..77B],
      width: [0..377B],
      bits: POINTER,
      height: CARDINAL
 ];
DCBHead: POINTER TO POINTER TO DCB = LOOPHOLE[420B];
currentDCB: POINTER TO DCB _ DCBHead^;
XOffset,YOffset: CARDINAL _ 0;
UNTIL currentDCB = NIL DO
IF currentDCB.bits = Screen.bits THEN EXIT;
YOffset _ YOffset + currentDCB.height*2;
currentDCB _ currentDCB.link;
ENDLOOP;

IF currentDCB = NIL THEN YOffset _ 0 ELSE XOffset _ currentDCB.offset*16;

index _ 177777B;
FOR i IN [0..nItems) DO
      BEGIN sLen: CARDINAL _ stringLen[itemList[i]];
      IF sLen > menuWidth THEN menuWidth _ sLen;
      END;
ENDLOOP;

GraphicsDefs.CreateBox[@menuBox,menuWidth,menuHeight];
GraphicsDefs.PutArea[0,0,0,menuBitmap.nLines-1,menuBitmap];
GraphicsDefs.PutArea[0,menuBitmap.nLines-1,
                  menuBitmap.nBits-1,menuBitmap.nLines-1,menuBitmap];
GraphicsDefs.PutArea[menuBitmap.nBits-1,menuBitmap.nLines-1,
                  menuBitmap.nBits-1,0,menuBitmap];
GraphicsDefs.PutArea[menuBitmap.nBits-1,0,0,0,menuBitmap];
GraphicsDefs.DisplayBox[@menuBox,CursorX^-XOffset,CursorY^-YOffset];

FOR i IN [0..nItems) DO
[]_GraphicsDefs.PutText[itemList[i],0,font.ascent+i*fontHeight,font,0,
                  menuBitmap];
ENDLOOP;
```

```
GraphicsDefs.PaintBox[@menuBox];

WHILE CursorX^-XOffset IN [MAX[20,b.displayX]-20..b.displayX+b.boxBitmap.nBits+20)
AND       CursorY^-YOffset IN [MAX[20,b.displayY]-20..b.displayY+b.boxBitmap.nLines+20) DO
      WHILE MouseButtons.Yellow = 0 DO
      GraphicsDefs.MoveBox[@menuBox,CursorX^-XOffset,CursorY^-YOffset];
      ENDLOOP;
      IF MouseButtons.Red = 0 THEN --selection
            BEGIN
            newIndex: CARDINAL _
                  MIN[nItems-1,(CursorY^-YOffset-menuBox.displayY)/fontHeight];
            IF newIndex = index THEN LOOP;
            IF index IN [0..nItems) THEN Select[index];
            index _ newIndex;
            Select[index];
            END;
ENDLOOP;

GraphicsDefs.DestroyBox[@menuBox,TRUE];
RETURN[index];
END;

lastBuffer: POINTER TO UNSPECIFIED _ NIL;
wordsLeft: CARDINAL _ 0;
InitReadBlock  : PUBLIC PROCEDURE =
BEGIN
wordsLeft _ 0;
END;

ReadBlock : PUBLIC PROCEDURE[sfd: POINTER,p: POINTER TO UNSPECIFIED,nwords: CARDINAL] RETURNS [POINTER TO
UNSPECIFIED] =
BEGIN
dsfd: POINTER TO DiscoDefs.SFD _ sfd;
fa: AltoFileDefs.FA;
wordsTransferred: CARDINAL _ 0;
thisTransfer: CARDINAL;

--IF nwords <= wordsLeft THEN
--      BEGIN
--      wordsLeft _ wordsLeft - nwords;
--      RETURN[lastBuffer+(256-(wordsLeft+nwords))];
--      END;
UNTIL nwords = 0 DO
      thisTransfer _ MIN[wordsLeft,nwords];
      InlineDefs.COPY[to: p+wordsTransferred,from: lastBuffer + 256 - wordsLeft,nwords:thisTransfer];
      nwords _ nwords - thisTransfer;
      wordsLeft _ wordsLeft - thisTransfer;
      wordsTransferred _ wordsTransferred + thisTransfer;
      IF wordsLeft = 0 THEN
            BEGIN
            lastBuffer _ DiscoDefs.ScanBuffer[dsfd,@fa];
            wordsLeft _ 256;
            END;
ENDLOOP;
wordsTransferred _ wordsTransferred + thisTransfer;
RETURN[p];
END;

END.
```

```
XMAllocDefs  : DEFINITIONS =
BEGIN

--routines exported by XMAlloc

AddToXMZone  : PROCEDURE [longHandle: LONG POINTER TO UNSPECIFIED,nwords: CARDINAL];

AllocateFromBank   : PROCEDURE [nwords: CARDINAL,bank: CARDINAL] RETURNS [LONG POINTER];

Allocate  : PROCEDURE [nwords: CARDINAL] RETURNS [LONG POINTER];

Free : PROCEDURE[LONG POINTER];

END.
```

```
DIRECTORY
      InlineDefs: FROM "InlineDefs",
      LongDefs: FROM "LongDefs",
      MiscDefs: FROM "MiscDefs",
      SegmentDefs: FROM "SegmentDefs",
      XMAllocDefs: FROM "XMAllocDefs";

XMAlloc : PROGRAM IMPORTS InlineDefs, LongDefs, MiscDefs, SegmentDefs EXPORTS XMAllocDefs =
BEGIN
touchCount: CARDINAL _ 0;
freeRejectCount: CARDINAL _ 0;
badHint: CARDINAL _ 0;
goodHint: CARDINAL _ 0;
MemBlockHeader: TYPE = RECORD
 [    link: LONG POINTER TO MemBlockHeader,
      length: CARDINAL,
      free: CARDINAL,
      freeHint: POINTER
 ];
DataHeaderStatus: TYPE = RECORD
 [ inUse,top,bottom: BOOLEAN
 ];
DataHeader: TYPE = RECORD
 [ status: DataHeaderStatus,
      end: POINTER TO DataTrailer
 ];
DataTrailer: TYPE = RECORD
 [ start: POINTER TO DataHeader,
      blockHead: POINTER TO MemBlockHeader
 ];

minPages: CARDINAL = 16;   --if alloc fails, mininum amount to add to zone
MemBlockOverhead: CARDINAL = SIZE[MemBlockHeader];
DataBlockOverhead: CARDINAL = SIZE[DataHeader]+SIZE[DataTrailer];
MinDataBlockSize: CARDINAL = 4 + DataBlockOverhead;
Root: ARRAY [1..3] OF LONG POINTER TO MemBlockHeader _ [NIL,NIL,NIL];

MakeLongPointer  : PROCEDURE [ptr: POINTER,bank: UNSPECIFIED] RETURNS [LONG POINTER] = MACHINE CODE BEGIN END;

AddToXMZone  : PUBLIC PROCEDURE [longHandle: LONG POINTER TO MemBlockHeader,nwords: CARDINAL] =
BEGIN
longDataH: LONG POINTER TO DataHeader _
                  LOOPHOLE[longHandle + SIZE[MemBlockHeader]];
longDataT: LONG POINTER TO DataTrailer _
                  LOOPHOLE[longHandle + (nwords - SIZE[DataTrailer])];
bank: CARDINAL _ InlineDefs.HighHalf[longHandle];
dhs: DataHeaderStatus _ [FALSE,TRUE,TRUE];

LongDefs.WriteLongPointer[Root[bank],@longHandle.link];
LongDefs.Write[nwords - MemBlockOverhead,@longHandle.length];
LongDefs.Write[nwords - MemBlockOverhead,@longHandle.free];
LongDefs.Write[InlineDefs.LowHalf[longDataH],@longHandle.freeHint];
LongDefs.Write[InlineDefs.LowHalf[longHandle],@longDataT.blockHead];
LongDefs.Write[dhs,@longDataH.status];
LongDefs.Write[InlineDefs.LowHalf[longDataT],@longDataH.end];
LongDefs.Write[InlineDefs.LowHalf[longDataH],@longDataT.start];

Root[bank] _ longHandle;
END;

Allocate : PUBLIC PROCEDURE [nwords: CARDINAL] RETURNS [LONG POINTER] =
BEGIN
maxFree: CARDINAL _ 0;
maxBank: CARDINAL;
longHandle: LONG POINTER TO MemBlockHeader;
i: CARDINAL;
FOR i IN [1..3] DO
      longHandle _ Root[i];
      UNTIL longHandle = NIL DO
      IF LongDefs.Read[@longHandle.free] > maxFree THEN
            BEGIN
            maxFree _ LongDefs.Read[@longHandle.free];
```

```
            maxBank _ i;
            END;
        longHandle _ LongDefs.ReadLongPointer[longHandle];
        ENDLOOP;
ENDLOOP;
IF maxFree<nwords THEN
        BEGIN
        additionalPages: CARDINAL _
            MAX[(nwords+MemBlockOverhead+DataBlockOverhead+255)/256,minPages];

        AddToXMZone[
            SegmentDefs.LongDataSegmentAddress[
                SegmentDefs.NewDataSegment[
                    SegmentDefs.DefaultXMBase,additionalPages]],additionalPages*256];
        RETURN[Allocate[nwords]];
        END;
RETURN[AllocateFromBank[nwords,maxBank]];
END;

AllocateFromBank   : PUBLIC PROCEDURE [nwords: CARDINAL,bank: CARDINAL] RETURNS [LONG POINTER] =
BEGIN
longHandle: LONG POINTER TO MemBlockHeader _ Root[bank];
handle: POINTER TO MemBlockHeader;
totalFree: CARDINAL _ 0;
head: POINTER TO DataHeader;
tail: POINTER TO DataTrailer;
blockSize: CARDINAL;
dhs: DataHeaderStatus;

UNTIL longHandle = NIL DO
        IF LongDefs.Read[@longHandle.free] >= nwords+DataBlockOverhead THEN
        BEGIN
        useHint: BOOLEAN _ FALSE;
        handle _ InlineDefs.LowHalf[longHandle];
        head _ LongDefs.Read[@longHandle.freeHint];
        IF head # NIL THEN
            BEGIN
            tail _ LongDefs.ReadBank[@head.end,bank];
            blockSize _ LOOPHOLE[tail+SIZE[DataTrailer],CARDINAL]-
                                    LOOPHOLE[head,CARDINAL];
            dhs _ LongDefs.ReadBank[@head.status,bank];
            IF NOT dhs.inUse THEN
                IF (blockSize-DataBlockOverhead)>=nwords THEN useHint _ TRUE
            ELSE LongDefs.Write[0,@longHandle.freeHint];
            END;
IF useHint THEN goodHint _ goodHint+1;
        IF NOT useHint THEN
            BEGIN
badHint _ badHint+1;
            head _ LOOPHOLE[handle + SIZE[MemBlockHeader]];
            tail _ LongDefs.ReadBank[@head.end,bank];
            END;
        --go through all blocks, looking for the right guy
        DO
        IF (head # LongDefs.ReadBank[@tail.start,bank]) OR
                (LongDefs.ReadBank[@head.end,bank] # tail) THEN
                    MiscDefs.CallDebugger["bad block encountered"];
        blockSize _ LOOPHOLE[tail+SIZE[DataTrailer],CARDINAL]-
                                    LOOPHOLE[head,CARDINAL];
        dhs _ LongDefs.ReadBank[@head.status,bank];
touchCount _ touchCount+1;
        IF NOT dhs.inUse THEN --free block
            BEGIN
            dataSize: CARDINAL _ blockSize - DataBlockOverhead;
            IF dataSize >= nwords THEN
                BEGIN
                --make data block out of new storage area
                IF dataSize >= nwords + MinDataBlockSize THEN --split
                    BEGIN
                    newTail: POINTER TO DataTrailer _
                                        LOOPHOLE[head + SIZE[DataHeader] + nwords];
                    newHead: POINTER TO DataHeader _
```

```
                                        LOOPHOLE[newTail + SIZE[DataTrailer]];
                    newDHS: DataHeaderStatus _ dhs;
                    dhs.bottom _ FALSE;
                    newDHS.top _ FALSE;
                    LongDefs.WriteBank[newTail,@head.end,bank];
                    LongDefs.WriteBank[head,@newTail.start,bank];
                    LongDefs.WriteBank[handle,@newTail.blockHead,bank];
                    LongDefs.WriteBank[tail,@newHead.end,bank];
                    LongDefs.WriteBank[newHead,@tail.start,bank];
                    LongDefs.WriteBank[newDHS,@newHead.status,bank];
                    tail _ newTail;
                    END;
                dhs.inUse _ TRUE;
                LongDefs.WriteBank[dhs,@head.status,bank];
                LongDefs.Write[CARDINAL[LongDefs.Read[@longHandle.free]] -
                    (LOOPHOLE[tail+SIZE[DataTrailer],CARDINAL]-
                        LOOPHOLE[head,CARDINAL]),@longHandle.free
                    ];
                --try your best to update freeHint
                tail _  LongDefs.ReadBank[@head.end,bank];
                IF NOT dhs.bottom THEN
                    LongDefs.Write[tail+SIZE[DataTrailer],@longHandle.freeHint];
                RETURN[MakeLongPointer[head+SIZE[DataHeader],bank]];
                END; --big enough
freeRejectCount _ freeRejectCount+1;
            END; --check free
        IF dhs.bottom THEN EXIT;
        head _ LOOPHOLE[tail + SIZE[DataTrailer]];
        tail _ LongDefs.ReadBank[@head.end,bank];
ENDLOOP; --end of checking all Mem Blocks
END;    --If fit is possible

longHandle _ LongDefs.ReadLongPointer[longHandle];

ENDLOOP;    --for all zone blocks

BEGIN
additionalPages: CARDINAL _
    MAX[(nwords+MemBlockOverhead+DataBlockOverhead+255)/256,minPages];

AddToXMZone[
    SegmentDefs.LongDataSegmentAddress[
        SegmentDefs.NewDataSegment[
            SegmentDefs.DefaultXMBase,additionalPages]],additionalPages*256];
RETURN[AllocateFromBank[nwords,bank]];
END;

END;


Free : PUBLIC PROCEDURE[finger: LONG POINTER] =
BEGIN
longHead: LONG POINTER TO DataHeader _ finger - SIZE[DataHeader];
bank: CARDINAL _ InlineDefs.HighHalf[longHead];
longTail: LONG POINTER TO DataTrailer _
        MakeLongPointer[LongDefs.Read[@longHead.end],bank];
longBlock: LONG POINTER TO MemBlockHeader _
    MakeLongPointer[LongDefs.Read[@longTail.blockHead],bank];
head: POINTER TO DataHeader _ InlineDefs.LowHalf[longHead];
tail: POINTER TO DataTrailer _ InlineDefs.LowHalf[longTail];
dhs: DataHeaderStatus _ LongDefs.Read[@longHead.status];

IF (head # LongDefs.Read[@longTail.start]) OR
    (NOT dhs.inUse) THEN
        MiscDefs.CallDebugger["trying to free bad block"];

--update the free count
LongDefs.Write[LongDefs.Read[@longBlock.free] +
        LOOPHOLE[tail + SIZE[DataTrailer],CARDINAL] - LOOPHOLE[head,CARDINAL],
        @longBlock.free];

dhs.inUse _ FALSE;
```

```
LongDefs.Write[dhs,@longHead.status];
LongDefs.Write[head,@longBlock.freeHint];
--merge adjacent free blocks....
IF NOT dhs.top THEN --try above
    BEGIN
    newTail: POINTER TO DataTrailer _ LOOPHOLE[head-SIZE[DataTrailer]];
    newHead: POINTER TO DataHeader _ LongDefs.ReadBank[@newTail.start,bank];
    ndhs: DataHeaderStatus _ LongDefs.ReadBank[@newHead.status,bank];
    IF NOT ndhs.inUse THEN
        BEGIN
        IF dhs.bottom THEN
            BEGIN
            ndhs.bottom _ TRUE;
            LongDefs.WriteBank[ndhs,@newHead.status,bank];
            END;
        dhs _ ndhs;
        head _ newHead;
        LongDefs.WriteBank[tail,@newHead.end,bank];
        LongDefs.WriteBank[newHead,@tail.start,bank];
        LongDefs.Write[newHead,@longBlock.freeHint];
        END;
    END;
IF NOT dhs.bottom THEN --and below
    BEGIN
    newHead: POINTER TO DataHeader _ LOOPHOLE[tail + SIZE[DataTrailer]];
    newTail: POINTER TO DataTrailer _ LongDefs.ReadBank[@newHead.end,bank];
    ndhs: DataHeaderStatus _ LongDefs.ReadBank[@newHead.status,bank];
    IF NOT ndhs.inUse THEN
        BEGIN
        IF ndhs.bottom THEN
            BEGIN
            dhs.bottom _ TRUE;
            LongDefs.WriteBank[dhs,@head.status,bank];
            END;
        LongDefs.WriteBank[newTail,@head.end,bank];
        LongDefs.WriteBank[head,@newTail.start,bank];
        END;
    END;

END;

END.
```