
Subject: Maker Interchange Format (MIF) device driver, v.1.06, Part01/02
Posted by [Marty Ryba](#) on Fri, 10 Oct 1997 07:00:00 GMT
[View Forum Message](#) <> [Reply to Message](#)

This is a multi-part message in MIME format.

-----7A83A0A1267

Content-Type: text/plain; charset=us-ascii

Content-Transfer-Encoding: 7bit

Here (in two parts) is code that allows IDL to create plots in Maker Interchange Format (MIF), an ASCII file format that is importable into FrameMaker(TM) from Adobe Systems, Inc. This version is a small bugfix over version 1.05 posted in April 1996; it should be the final version because it fixes the last little annoying bug: previous versions had problems with images (from TV) that had an odd number of pixels per row.

There are two versions of the source, one for IDL version 3.6 and earlier, and one for version 4.0 and later (verified to work under IDL 5.0.2). Makefile instructions for SunOS 4.1.x, 5.x (Solaris 2.x), SGI IRIX 5.2, and HPUX (non-functional?) are included; since there is nothing very special in the code it should compile fine on other platforms/compilers that support sharable library creation. If you compile this code on another platform, send the Makefile to me and I'll include the flags for future releases (if any).

A sample procedure, loadmif.pro, is included to facilitate automatic loading of the proper shareable object given IDL version and OS release type. It may need customization for specifics of your installation.

The driver supports nearly all constructs possible under the PostScript driver (fonts, etc.) and FrameMaker features like spot color. See the README for further information. This driver would make a good template for creation of one for Adobe Illustrator...anyone game?

--

Dr. Marty Ryba | Of course nothing I say here is official
MIT Lincoln Laboratory | policy, and Laboratory affiliation is
ryba@ll.mit.edu | for identification purposes only,
 | blah, blah, blah, ...

-----7A83A0A1267

Content-Type: text/plain; charset=us-ascii; name="Part01"

Content-Transfer-Encoding: 7bit

Content-Disposition: inline; filename="Part01"

#!/bin/sh

This is a shell archive. Remove anything before this line, then unpack

```

# it by saving it into a file and typing "sh file". To overwrite existing
# files, type "sh file -c". You can also feed this as standard input via
# unshar, or by typing "sh <file", e.g.. If this archive is complete, you
# will see the following message at the end:
# "End of archive 1 (of 2)."
```

Contents: README Makefile frameimage.h mif_3.6.c

Wrapped by ryba@skeleton on Fri Oct 10 10:25:37 1997

```

PATH=/bin:/usr/bin:/usr/ucb ; export PATH
if test -f 'README' -a "${1}" != "-c" ; then
    echo shar: Will not clobber existing file \"README\"
else
    echo shar: Extracting \"README\" (6491 characters)
    sed "s/^X//" >'README' <<'END_OF_FILE'
X
X IDL device driver for Maker Interchange Format (MIF)
X
X Version 1.06 - 10 October 1997
X
X Martin Ryba, MIT Lincoln Laboratory
X ryba@ll.mit.edu
X
X
X IDL can link in routines from sharable object libraries via the
XLINKIMAGE command. In this manner, new graphics device drivers can
Xalso be added. This is a driver for Maker Interchange Format, an
XASCII format used by FrameMaker(TM) from Adobe Systems, Inc. Using
Xthe MIF driver allows one to create an IDL graph that becomes an
Xeditable part of a FrameMaker document, allowing one to change the
Xproperties (colors, line styles, fonts, tick marks, etc.) of the
Xgraphic without re-running the IDL .pro file that generated it. Also,
Xthe graphic elements position and size are visible at full resolution,
Xunlike the low-res monochrome bitmaps available under the Encapsulated
XPostScript Interchange (EPSI) format driver.
X
XTo use the driver within IDL:
XIDL> linkimage, 'mif_dev', 'mif.so', /device
XIDL> set_plot, 'mif'
X
XReplace 'mif.so' with the full pathname of the shareable object
Xlibrary. The linkimage command is of course easily added to your
Xstartup file.
X
X This driver supports nearly all the keywords and features of the
XPostScript driver with IDL. The default line thickness is 1pt (as
Xopposed to the too-thin ~9/32 pt used by the PS driver). The default
Xgraphic size is 6.5"x6.5", though that can be changed via the
X{X,Y}SIZE keywords. There are no offset keywords since these MIF files
Xare meant to be incorporated into other documents. If they are simply
```

Xopened and printed in FrameMaker, the graphic will be centered on the Xpage with landscape or portrait automatically chosen depending on the Xdimensions of the plot. Color images (scalable pixels) are supported, Xand lines and polygons can be created in any of the eight FrameMaker Xstandard spot colors: black, white, red, green, blue, cyan, magenta, Xand yellow. IDL linestyles are supported, and polygon fill patterns Xvia the FILL_PATTERN keyword of POLYFILL are supported (FrameMaker Xsupplies 16 fill patterns, though fill 15 (none) isn't very useful in Xthis case).

X

X A note on dashed line styles...since I don't have any logic in Xthe driver to measure how far along a given line I've traveled, I've Xnoticed that if the points are closer together the dashes won't show Xup, since FrameMaker wants to show each vertex of a PolyLine, even for Xa dashed line style. I've found that selecting the object in Frame Xand then choosing the "Smooth" option from the Graphics menu will make Xthe dashes show up without too much change to the nature of the line.

X

X All 35 standard PostScript fonts are supported, as well as the X!-escapes for changing fonts; font !20 can still be an arbitrary Xuser-defined font. Superscripts and Subscripts are supported via the X!U and !D escapes (other subscript sizes and levels (!A, !B, !E, !I, Xand !L) are not supported). Save and restore of positions (!S and !R) Xare not supported. The !MX bullet is not supported; instead use !B or Xstring(165b). The character set is the Frame Character set described Xin Appendix B of the User's Guide. Below is a list of the supported Xkeywords of the DEVICE procedure and their functions:

X

XAVANTGARDE - select ITC Avant Garde PostScript font

XBKMAN - select ITC Bookman PostScript font

XBOLD - select bold version of current font (if available)

XBOOK - select book version of current font (if available)

XCLOSE_FILE - close current MIF file. Resets the file name to the

X default idl.mif

XCOURIER - select Courier PostScript font

XDEMI - select demibold version of current font (if avail.)

XFILENAME - set the current MIF file name

XFONT_INDEX - see PS driver description (3-26 of Ref. manual)

XFONT_SIZE - set default font size (normally 12pt)

XHELVETICA - select Helvetica PostScript font

XINCHES - {X,Y}SIZE values are in inches

XITALIC - select italic version of current font (if avail.)

XLIGHT - select light version of current font (if available)

XMEDIUM - select medium version of current font (if avail.)

XNARROW - select narrow version of current font (if avail.)

XOBLIQUE - select oblique version of current font (if avail.)

XPALATINO - select Palatino PostScript font

XPOINTS - {X,Y}SIZE values are in points (1/72 inch)

XSCHOOLBOOK - select New Century Schoolbook PostScript font
 XSYMBOL - select Symbol PostScript font
 XTIMES - select Times Roman PostScript font
 XUSER_FONT - scalar string containing name of a PostScript font.
 X Font must be specified exactly as PS printer
 X expects. May require some FrameMaker setup.
 XXSIZE, YSIZE - Set size of drawing area. Default is cm, but see
 X INCHES and POINTS keywords.
 XZAPFCHANCERY - select ITC Zapf Chancery PostScript font
 XZAPFDINGBATS - select ITC Zapf Dingbats PostScript font
 X
 X
 X Notes on installing:
 X
 X1) The Makefile contains compiler options for several platforms. I
 X have tested it on SunOS 4.1.x and SunOS 5.5 (Solaris 2.5). Outside
 X contributions for SGI and HPUX are provided, though the HPUX one
 X may not work yet. I haven't needed code changes (so far); VMS, Mac,
 X and Windows versions would need work.
 X
 X Comments, suggestions, bug reports, bug fixes, etc. are welcome,
 X though they will be acted upon as I'm able.
 X
 X
 X**** CHANGES *****
 XVersion 1.06 - 10/10/97 - Quashed bug that caused images with an odd
 X number of pixels to not be read properly; turns out images must
 X be padded to an even number of bytes *per line*. Also added
 X carriage-return to all printf() statements; IDL must have re-
 X defined printf() to make it compatible across platforms.
 X
 XVersion 1.05 - 04/11/96 - Finally really quashed bug in upside-down
 X images...turns out Angle is an object property *not* inherited
 X from earlier objects (always defaults to zero).
 X
 XVersion 1.04 - 02/09/96 - Added separate source file for IDL 4.0,
 X since export.h is significantly different. Found and quashed
 X bug in images--they came out upside down before! (Thanks
 X Dan). The Makefile is improved.
 X
 XVersion 1.03 - 01/30/96 - Finally report out text width (man, what a pain!).
 X Requires Environment variable FMHOME to be defined; not sure of
 X portability problems with that...may need ifdef's in code to handle
 X other OS's. Changed resolution to 0.2pt to reduce need for extra
 X decimal place in coords.
 X
 XVersion 1.02 - 01/11/96 - Don't add point in continuing line if start and
 X stop points are identical. FONT_SIZE bugfix.

```

X
XVersion 1.01 - 07/10/95 - Small bugfixes for fill, page number, !-escape
X handling, and added debugging prints.
X
END_OF_FILE
if test 6491 -ne `wc -c <'README'`; then
    echo shar: \"README\" unpacked with wrong size!
fi
# end of 'README'
fi
if test -f 'Makefile' -a "${1}" != "-c" ; then
    echo shar: Will not clobber existing file \"Makefile\"
else
    echo shar: Extracting \"Makefile\" \ (1255 characters\ )
    sed "s/^X//\" >'Makefile' <<'END_OF_FILE'
X# Makefile for shared library for IDL LINKIMAGE for MIF device driver
X
X# Set IDL_VERSION to correspond...3.6 for 3.x and 4.0 for 4.x
X# The location of your idl directory should be IDL_DIR
X# For 3.x export.h is in IDL_DIR/source; in 4.x now in IDL_DIR/external
XIDL_VERSION = 4.0
XIDL_DIR = /usr/local/rsi/idl_4
X#INC = ${IDL_DIR}/source
XINC = ${IDL_DIR}/external
X
XSRC = mif_${IDL_VERSION}.c
X
XOBJ = mif_${IDL_VERSION}.o
X
X# SGI and SunOS
XTARGET = mif_${IDL_VERSION}.so
X# HPUX
X#TARGET = mif_${IDL_VERSION}.sl
X
X
X# These are for SunOS 4.1.3 using the SunPro compiler
XCFLAGS = -vc -Xc -pic -fsingle -O -I$(INC)
XCC = acc
XLDFLAGS = -L/usr/local/lang_3.0/SC2.0.1
XLIBS = -assert nosymbolic -Bstatic -lansi
X
X# These are for Solaris 2.x using the SunPro compiler
X# CFLAGS = -vc -Xc -Kpic -O -I$(INC)
X# CC = cc
X# LDFLAGS = -G
X# LIBS =
X
X# Flags for a SGI IRIX 5.2

```

```

X# CFLAGS = -xansi -KPIC -I$(INC)
X# CC = cc
X# LD = ld
X# LDFLAGS = -shared
X# LIBS =
X
X# Flags for HPUX using the HP compiler (ng so far though)
X# CFLAGS = -c +z -Aa -I$(INC)
X# CC = cc
X# LD = ld
X# LDFLAGS = -b
X# LIBS =
X
X$(TARGET): $(OBJ)
X $(LD) -o $(TARGET) $(LDFLAGS) $(OBJ) $(LIBS)
X
Xclean:
X rm -f $(OBJ) $(TARGET) core
X
Xshar:
X makekit -p -s56k README Makefile frameimage.h \
X mif_3.6.c loadmif.pro mif_4.0.c
END_OF_FILE
if test 1255 -ne `wc -c <'Makefile'`; then
    echo shar: \"Makefile\" unpacked with wrong size!
fi
# end of 'Makefile'
fi
if test -f 'frameimage.h' -a "${1}" != "-c" ; then
    echo shar: Will not clobber existing file \"frameimage.h\"
else
    echo shar: Extracting \"frameimage.h\" \ (507 characters)
    sed "s/^X//\" >'frameimage.h' <<'END_OF_FILE'
X/*
X * FrameImage.h - Description of FrameMaker(TM) raster format
X *
X *      From "Integrating Applications with FrameMaker"
X *      Appendix A
X */
X
Xtypedef int IntT;
X
Xtypedef struct {
X  IntT ras_magic;
X  IntT ras_width;
X  IntT ras_height;
X  IntT ras_depth;
X  IntT ras_length;

```

```

X  IntT ras_type;
X  IntT ras_maptype;
X  IntT ras_maplength;
X} RasterfileT;
X
X#define RAS_MAGIC 0x59a66a95
X#define RT_STANDARD 1
X#define RT_BYTE_ENCODED 2
X#define RMT_NONE 0
X#define RMT_EQUAL_RGB 1
END_OF_FILE
if test 507 -ne `wc -c <'frameimage.h`; then
    echo shar: \"frameimage.h\" unpacked with wrong size!
fi
# end of 'frameimage.h'
fi
if test -f 'mif_3.6.c' -a "${1}" != "-c" ; then
    echo shar: Will not clobber existing file \"mif_3.6.c\"
else
    echo shar: Extracting \"mif_3.6.c\" (46780 characters)
    sed "s/^X//\" >'mif_3.6.c' <<'END_OF_FILE'
X/*****
X *
X *
X * mif.c - a MIF ((Frame)Maker Interchange Format) graphics device driver *
X *      for IDL (Interactive Data Language) *
X *
X *      May 1995, M.F. Ryba, F.K. Knight, MIT Lincoln Laboratory *
X *
X *      With sincere thanks to J. Stillerman, MIT Plasma Fusion Center *
X *      for his QMS device driver as a template *
X *
X * To use: *
X * IDL> linkimage, 'mif_dev', 'mif.so', /device *
X * IDL> set_plot, 'mif' *
X *
X *****/
X
X#include <stdio.h>
X#include <stdlib.h>
X#include <string.h>
X
X#define unix /* Dunno why... */
X#include "export.h"
X
X#include "frameimage.h" /* Description of FrameImage format */
X
Xtypedef UCHAR BOOL; /* Boolean flags */
X

```

```

X/* For debugging printouts, uncomment this */
X/* #define DEBUG */
X
X/*
X * Procedure declarations
X */
Xstatic void mif_draw(GR_PT *, GR_PT *, ATTR_STRUCT *);
Xstatic int mif_text(GR_PT *, ATTR_STRUCT *, TEXT_STRUCT *, char *);
Xstatic void mif_erase(ATTR_STRUCT *);
Xstatic void mif_poly(int *, int *, int, POLYFILL_ATTR *);
Xstatic void mif_color(long, long);
Xstatic void mif_image(UCHAR *, int, int, int, int, int, TV_STRUCT *);
Xstatic void mif_device(int, VPTR *, char *);
Xstatic void mif_help(int, VPTR *);
X
Xstatic void mif_openfile(void);
Xstatic void mif_prepfiler(void);
Xstatic void mif_closefile(void);
Xstatic void mif_linestyle(int);
Xstatic void mif_font(void);
Xstatic void end_str(BOOL *);
Xstatic void do_char(UCHAR, BOOL *, int *, BOOL);
Xstatic void write_hex(UCHAR *, int);
Xstatic int find_bit(long);
X
X/*
X * For making readable output
X */
X#define LINE_BREAK 75 /* # chars on line before line break */
X#define LINE_LEN 256 /* max # chars per output line */
X
X/*
X * Defines for some default behaviours, device size, etc.
X * For now we will use resolution of 1/5 pt (72pt/inch)
X */
X#define CM_PER_INCH 2.54 /* # of centimeters in an inch */
X#define PIXELS_INCH 360 /* # of IDL pixels mapped into an inch */
X#define PIXELS_CM 142 /* # of IDL pixels per cm */
X#define DEF_XSIZE 2340 /* Default to 6.5inx6.5in square area */
X#define DEF_YSIZE 2340
X#define DEF_XS_PT 468.0f /* This size in points */
X#define DEF_YS_PT 468.0f /* This size in points */
X#define DEF_XCSIZE 35 /* Default X,Y character sizes */
X#define DEF_YCSIZE 60 /* (about 12pt) */
X#define CTAB_SIZE 256 /* Size of color table */
X
X/*
X * Global Variables

```



```

X */
Xstatic const char *mif_version = "1.05 11 April 1996";
X
Xglobal DEVICE_DEF mif_dev = { /* Definition for MIF device */
X {3,0,"MIF"}, /* STRING for name */
X {DEF_XSIZE, DEF_YSIZE}, /* Total size in device coordinates */
X {DEF_XSIZE, DEF_YSIZE}, /* Visible area size, device coords */
X {DEF_XCSIZE, DEF_YCSIZE}, /* Default character sizes */
X {PIXELS_CM, PIXELS_CM}, /* Device units / centimeter, x & y. */
X CTAB_SIZE, /* # of possible simultaneous colors */
X CTAB_SIZE, /* Size of color table */
X 1, /* minimum line spacing for solid fill,
X ignored if hardware fill present. */
X -1, /* Current window number */
X 0, /* Unit number of output file */
X /* Advertise limitations and abilities */
X D_SCALABLE_PIXELS|D_ANGLE_TEXT|D_THICK|D_IMAGE|D_COLOR|D_POL YFILL|
X D_WHITE_BACKGROUND|D_HERSH_CONTROL,
X { 0, 0}, /* Display XY origin */
X { 1, 1}, /* Display XY zoom */
X 1.0f, /* Aspect ratio = v_size[0] / v_size[1] */
X { /* Core routine pointers */
X mif_draw, /* ^ to draw routine */
X mif_text, /* text output, or NULL */
X mif_erase, /* erase */
X NULL, /* cursor inquire and set, or NULL */
X mif_poly, /* Fill irregular polygon, or NULL */
X NULL, /* Return to interactive mode, or NULL */
X NULL, /* Flush entry, or NULL */
X mif_color, /* Load color tables, or NULL */
X mif_image, /* Pixel input/output, or NULL */
X mif_device, /* Rout. to call from DEVICE proc, or NULL.*/
X mif_help, /* Rout. to call for HELP,/DEVICE, or NULL */
X NULL /* Routine called when loaded */
X },
X { /* Window system routines */
X NULL, /* Create a window */
X NULL, /* Delete a window */
X NULL, /* Expose a window */
X NULL, /* Set the current window */
X NULL, /* Menu function */
X },
X (char *) NULL
X};
X
X/*
X * Local static variables (state info, etc.)
X */

```

```

Xstatic char pout_buf[LINE_LEN]; /* Buffer for character output */
X
X/*
X * Structures for font characteristics. There are 35 standard
X * PostScript fonts
X */
X#define N_PS_FONTS 35
Xtypedef enum { /* Font families */
X  COURIER, /* Courier */
X  HELVETICA, /* Helvetica */
X  AVANTGARDE, /* ITC Avant Garde Gothic */
X  BOOKMAN, /* ITC Bookman */
X  ZAPF_CHANCERY, /* ITC Zapf Chancery */
X  ZAPF_DINGBAT, /* ITC Zapf Dingbats */
X  SCHOOLBOOK, /* New Century Schoolbook */
X  PALATINO, /* Palatino */
X  SYMBOL, /* Symbol */
X  TIMES /* Times */
X} TEXT_FFAM;
Xstatic char *family_names[] = { /* FrameMaker names for families */
X  "Courier", "Helvetica", "AvantGarde",
X  "Bookman", "ZapfChancery", "ZapfDingbats",
X  "NewCenturySchlbk", "Palatino", "Symbol",
X  "Times"
X};
X
Xtypedef enum { /* Weight */
X  MEDIUM, /* Regular (Medium) */
X  BOLD, /* Bold */
X  LIGHT, /* Light */
X  BOOK, /* Book */
X  DEMI /* Demi */
X} TEXT_FWGT;
Xstatic char *weight_names[] = { /* Names for font weights */
X  "Regular", "Bold", "Light", "Book", "DemiBold"
X};
X
Xtypedef enum { /* Angle */
X  REGULAR, /* Regular */
X  ITALIC, /* Italic */
X  OBLIQUE /* Oblique */
X} TEXT_FANG;
Xstatic char *angle_names[] = { /* Names for font angles */
X  "Regular", "Italic", "Oblique"
X};
X
Xtypedef enum { /* Variation (for Helvetica) */
X  NORMAL, /* Regular */

```

```

X  NARROW  /* Narrow */
X} TEXT_FVAR;
Xstatic char *variation_names[] = { /* Names for font variations */
X  "Regular", "Narrow"
X};
X
Xtypedef struct {
X  TEXT_FFAM family;
X  TEXT_FWGT weight;
X  TEXT_FANG angle;
X  TEXT_FVAR variation;
X  char *name;
X  char *afm_name;
X} PS_FONT_T;
X
Xstatic PS_FONT_T psfonts[N_PS_FONTS] = {
X  { COURIER, MEDIUM, REGULAR, NORMAL, "Courier", "Cr" },
X  { COURIER, BOLD, REGULAR, NORMAL, "Courier Bold", "Cr-B" },
X  { COURIER, MEDIUM, OBLIQUE, NORMAL, "Courier Oblique", "Cr-O" },
X  { COURIER, BOLD, OBLIQUE, NORMAL, "Courier Bold Oblique", "Cr-BO" },
X  { HELVETICA, MEDIUM, REGULAR, NORMAL, "Helvetica", "H" },
X  { HELVETICA, BOLD, REGULAR, NORMAL, "Helvetica Bold", "H-B" },
X  { HELVETICA, MEDIUM, OBLIQUE, NORMAL, "Helvetica Oblique", "H-O" },
X  { HELVETICA, BOLD, OBLIQUE, NORMAL, "Helvetica Bold Oblique", "H-BO" },
X  { HELVETICA, MEDIUM, REGULAR, NARROW, "Helvetica Narrow", "H-N" },
X  { HELVETICA, BOLD, REGULAR, NARROW, "Helvetica Narrow Bold", "H-N-B" },
X  { HELVETICA, MEDIUM, OBLIQUE, NARROW, "Helvetica Narrow Oblique",
X  "H-N-O" },
X  { HELVETICA, BOLD, OBLIQUE, NARROW, "Helvetica Narrow Bold Oblique",
X  "H-N-BO" },
X  { AVANTGARDE, BOOK, REGULAR, NORMAL,
X  "ITC Avant Garde Gothic Book", "AG-Bk" },
X  { AVANTGARDE, BOOK, OBLIQUE, NORMAL,
X  "ITC Avant Garde Gothic Book Oblique", "AG-BkO" },
X  { AVANTGARDE, DEMI, REGULAR, NORMAL,
X  "ITC Avant Garde Gothic Demi", "AG-Dm" },
X  { AVANTGARDE, DEMI, OBLIQUE, NORMAL,
X  "ITC Avant Garde Gothic Demi Oblique", "AG-DmO" },
X  { BOOKMAN, DEMI, REGULAR, NORMAL, "ITC Bookman Demi", "BM-Dm" },
X  { BOOKMAN, DEMI, ITALIC, NORMAL, "ITC Bookman Demi Italic", "BM-DmI" },
X  { BOOKMAN, LIGHT, REGULAR, NORMAL, "ITC Bookman Light", "BM-L" },
X  { BOOKMAN, LIGHT, ITALIC, NORMAL, "ITC Bookman Light Italic", "BM-LI" },
X  { ZAPF_CHANCERY, MEDIUM, ITALIC, NORMAL,
X  "ITC Zapf Chancery Medium Italic", "ZC-MI" },
X  { ZAPF_DINGBAT, MEDIUM, REGULAR, NORMAL, "ITC Zapf Dingbats", "ZD" },
X  { SCHOOLBOOK, MEDIUM, REGULAR, NORMAL, "New Century Schoolbook",
X  "NCS-R" },
X  { SCHOOLBOOK, BOLD, REGULAR, NORMAL, "New Century Schoolbook Bold",

```

```

X "NCS-B" },
X { SCHOOLBOOK, MEDIUM, ITALIC, NORMAL,
X "New Century Schoolbook Italic", "NCS-I" },
X { SCHOOLBOOK, BOLD, ITALIC, NORMAL,
X "New Century Schoolbook Bold Italic", "NCS-BI" },
X { PALATINO, MEDIUM, REGULAR, NORMAL, "Palatino", "Pal-R" },
X { PALATINO, BOLD, REGULAR, NORMAL, "Palatino Bold", "Pal-B" },
X { PALATINO, MEDIUM, ITALIC, NORMAL, "Palatino Italic", "Pal-I" },
X { PALATINO, BOLD, ITALIC, NORMAL, "Palatino Bold Italic", "Pal-BI" },
X { SYMBOL, MEDIUM, REGULAR, NORMAL, "Symbol", "Symbol" },
X { TIMES, MEDIUM, REGULAR, NORMAL, "Times", "Tim-R" },
X { TIMES, BOLD, REGULAR, NORMAL, "Times Bold", "Tim-B" },
X { TIMES, MEDIUM, ITALIC, NORMAL, "Times Italic", "Tim-I" },
X { TIMES, BOLD, ITALIC, NORMAL, "Times Bold Italic", "Tim-BI" },
X};
X/*
X * FrameMaker character set (for fonts other than Symbol and Dingbats)
X */
Xstatic char *Encoding[256] = {
X  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",
X  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",
X  "", "", "", "", "space", "exclam", "quotedbl", "numbersign",
X  "dollar", "percent", "ampersand", "quotesingle", "parenleft",
X  "parenright", "asterisk", "plus", "comma", "hyphen", "period",
X  "slash", "zero", "one", "two", "three", "four", "five", "six",
X  "seven", "eight", "nine", "colon", "semicolon", "less", "equal",
X  "greater", "question", "at", "A", "B", "C", "D", "E", "F", "G",
X  "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T",
X  "U", "V", "W", "X", "Y", "Z", "bracketleft", "backslash",
X  "bracketright", "asciicircum", "underscore", "grave", "a", "b",
X  "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m", "n", "o",
X  "p", "q", "r", "s", "t", "u", "v", "w", "x", "y", "z", "braceleft",
X  "bar", "braceright", "asciitilde", "", "Adieresis", "Aring",
X  "Ccedilla", "Eacute", "Ntilde", "Odieresis", "Udieresis", "aacute",
X  "agrave", "acircumflex", "adieresis", "atilde", "aring", "ccedilla",
X  "eacute", "egrave", "ecircumflex", "edieresis", "iacute", "igrave",
X  "icircumflex", "idieresis", "ntilde", "oacute", "ograve", "ocircumflex",
X  "odieresis", "otilde", "uacute", "ugrave", "ucircumflex", "udieresis",
X  "dagger", "", "cent", "sterling", "section", "bullet", "paragraph",
X  "germandbls", "registered", "copyright", "trademark", "acute",
X  "dieresis", "", "AE", "Oslash", "", "", "", "", "yen", "", "", "", "",
X  "", "", "ordfeminine", "ordmasculine", "", "ae", "oslash",
X  "questiondown", "exclamdown", "logicalnot", "", "florin", "", "",
X  "guillemotleft", "guillemotright", "ellipsis", "", "Agrave", "Atilde",
X  "Otilde", "OE", "oe", "endash", "emdash", "quotedblleft",
X  "quotedblright", "quoteleft", "quoteright", "", "", "ydieresis",
X  "Ydieresis", "fraction", "currency", "guilsinglleft", "guilsinglright",
X  "fi", "fl", "daggerdbl", "periodcentered", "quotesinglbase",

```

```

X  "quotedblbase", "perthousand", "Acircumflex", "Ecircumflex", "Aacute",
X  "Edieresis", "Egrave", "Iacute", "Icircumflex", "Idieresis", "Igrave",
X  "Oacute", "Ocircumflex", "", "Ograve", "Uacute", "Ucircumflex",
X  "Ugrave", "dotlessi", "circumflex", "tilde", "macron", "breve",
X  "dotaccent", "ring", "cedilla", "hungarumlaut", "ogonek", "caron"
X};
X/*
X * This struct defines internal state of MIF driver
X */
Xstatic struct {
X  struct { /* Overall state of driver */
X  BOOL fopen; /* If file is open */
X  char fname[MAX_PATH_LEN]; /* File name (def: idl.mif) */
X  BOOL not_blank; /* Page is not blank */
X  VARIABLE lun; /* File LUN for output */
X  int cur_page; /* Current output page */
X  } state;
X  struct { /* Plotting region parms */
X  float width,height; /* Width height (points) */
X  int x,y; /* Width, height (pixels) */
X  } size;
X  struct { /* Text attributes */
X  BOOL newfont; /* New font (via device) */
X  int size; /* Font default size */
X  int psfont; /* PostScript font number */
X  int cursize; /* Current text size */
X  short wx[256], /* Font metric info */
X    llx[256],urx[256];
X  } ta;
X  struct { /* Current graphics attributes */
X  int color; /* Current line color */
X  float thick; /* Current line thickness */
X  int linestyle; /* Current linestyle */
X  int pen; /* Current pen style */
X  int fill; /* Current fill style */
X  BOOL in_line; /* In middle of polyline */
X  } ga;
X  struct { /* Color attributes */
X  int black; /* IDL index of black */
X  int white; /* IDL index of white */
X  int red; /* IDL index of red */
X  int green; /* IDL index of green */
X  int blue; /* IDL index of blue */
X  int cyan; /* IDL index of cyan */
X  int magenta; /* IDL index of magenta */
X  int yellow; /* IDL index of yellow */
X  long start,n; /* IDL color table vars */
X  } ca;

```

```

X  POUT_CNTRL pout; /* Control structure for pout() */
X} mifstat = {
X  {
X FALSE, /* File is closed */
X "idl.mif", /* Default file name */
X FALSE, /* Page is blank */
X {TYP_LONG, 0,}, /* No file LUN yet */
X 1, /* Initially page 1 */
X },
X {
X DEF_XS_PT, DEF_YS_PT, /* Width, Height */
X DEF_XSIZE, DEF_YSIZE, /* X, Y sizes */
X },
X {
X TRUE, /* New font asked for */
X 12, /* Default 12pt text */
X 4, /* Default Helvetica */
X 0, /* No size is current */
X },
X {
X -1, -1.0f, -1, /* Initially some undefined */
X 0, /* Pen (solid) */
X -1, /* Fill (undef) */
X FALSE, /* Not in mid-polyline */
X },
X {
X 0, CTAB_SIZE-1, -1, -1, /* Initially black is 0, white */
X -1, -1, -1, -1, /* is !d.n_colors-1 */
X 0, 0,
X },
X {
X 0, /* LUN for output file */
X 0, /* Current column */
X LINE_BREAK, /* Desired line break */
X (char *) NULL, /* No leading text */
X 0, /* So it has zero length */
X pout_buf, /* The output buffer */
X LINE_LEN /* Its length */
X },
X};
X
X/*
X * List of font names for ! escapes
X */
X#define NUM_FONTS 18 /* How many fonts I support */
X#define MAX_FONT_NAME_LEN 128 /* Because font names can change */
Xstatic struct mif_font {
X  char name[MAX_FONT_NAME_LEN];

```

```

X  int ps_font_num;
X} miffonts[NUM_FONTS] = {
X  { "Helvetica", 4 },
X  { "Helvetica-Bold", 5},
X  { "Helvetica-Narrow", 8 },
X  { "Helvetica-Narrow-BoldOblique", 11 },
X  { "Times-Roman", 31 },
X  { "Times-BoldItalic", 34 },
X  { "Symbol", 30 },
X  { "ZapfDingbats", 21 },
X  { "Courier", 0 },
X  { "Courier-Oblique", 2 },
X  { "Palatino-Roman", 26 },
X  { "Palatino-Italic", 28 },
X  { "Palatino-Bold", 27 },
X  { "Palatino-BoldItalic", 29 },
X  { "AvantGarde-Book", 12 },
X  { "NewCenturySchlbk-Roman", 22 },
X  { "NewCenturySchlbk-Bold", 23 },
X  { "<Undefined-User-Font>", -1 }
X};
X
X/*****
X *
X * Start of executable programs
X *
X *****/
X
X#define PREPARE if (!mifstat.state.not_blank) mif_prepfiler()
X#define END_LAST_LINE if(mifstat.ga.in_line) {\
X  pout(&mifstat.pout, POUT_FL, ">");\
X  mifstat.ga.in_line=FALSE;}
X
Xstatic void
Xmif_draw(GR_PT *p0, GR_PT *p1, ATTR_STRUCT *a)
X{
X  float x,y; /* Location of each point */
X  static GR_PT p1last; /* Attributes of last segment */
X
X  PREPARE;
X  if (a->thick == 0.0f) a->thick = 1.0f; /* Default thickness */
X/*
X * Check for continuation of previous line
X */
X  if (mifstat.ga.in_line && (p0->i.x == p1last.i.x) &&
X (p0->i.y == p1last.i.y) && (a->thick == mifstat.ga.thick) &&
X (a->color == mifstat.ga.color) &&
X (a->linestyle == mifstat.ga.linestyle)) {

```

```

X/*
X * Just place the new point. IDL uses lower left as origin, FrameMaker
X * uses the upper left. Also, if second point is in same location as
X * the first point, then skip it altogether.
X */
X if (p0->i.x != p1->i.x || p0->i.y != p1->i.y) {
X   x = (float) p1->i.x / 5.0f;
X   y = (float) (mifstat.size.y - p1->i.y) / 5.0f;
X   pout(&mifstat.pout, 0, "<Point %.1f %.1f>", x, y);
X }
X } else {
X END_LAST_LINE; /* End previous line */
X pout(&mifstat.pout, POUT_SL, "<PolyLine>");
X if (mifstat.ga.pen != 0) { /* Pen style */
X   pout(&mifstat.pout, 0, "<Pen 0>");
X   mifstat.ga.pen = 0;
X }
X if (mifstat.ga.fill != 15) { /* Fill style */
X   pout(&mifstat.pout, 0, "<Fill 15>");
X   mifstat.ga.fill = 15;
X }
X if (a->thick != mifstat.ga.thick) { /* Thickness change */
X   pout(&mifstat.pout, 0, "<PenWidth %.2f>", a->thick);
X   mifstat.ga.thick = a->thick;
X }
X if (a->color != mifstat.ga.color) { /* Color change */
X   mifstat.ga.color = a->color;
X   if (a->color == mifstat.ca.black) /* Spot Color */
X     pout(&mifstat.pout, 0, "<ObColor `Black'>");
X   else if (a->color == mifstat.ca.white)
X     pout(&mifstat.pout, 0, "<ObColor `White'>");
X   else if (a->color == mifstat.ca.red)
X     pout(&mifstat.pout, 0, "<ObColor `Red'>");
X   else if (a->color == mifstat.ca.green)
X     pout(&mifstat.pout, 0, "<ObColor `Green'>");
X   else if (a->color == mifstat.ca.blue)
X     pout(&mifstat.pout, 0, "<ObColor `Blue'>");
X   else if (a->color == mifstat.ca.cyan)
X     pout(&mifstat.pout, 0, "<ObColor `Cyan'>");
X   else if (a->color == mifstat.ca.magenta)
X     pout(&mifstat.pout, 0, "<ObColor `Magenta'>");
X   else if (a->color == mifstat.ca.yellow)
X     pout(&mifstat.pout, 0, "<ObColor `Yellow'>");
X   else {
X     printf("Line color cannot be spot color, using black\n");
X     pout(&mifstat.pout, 0, "<ObColor `Black'>");
X     mifstat.ga.color = mifstat.ca.black;
X   }

```



```

X }
X if (a->linestyle != mifstat.ga.linestyle) { /* Linestyle change */
X   mifstat.ga.linestyle = a->linestyle;
X   mif_linestyle(a->linestyle);
X }
X/*
X * Now place the points. IDL uses lower left as origin, FrameMaker
X * uses the upper left
X */
X x = (float) p0->i.x / 5.0f;
X y = (float) (mifstat.size.y - p0->i.y) / 5.0f;
X pout(&mifstat.pout, 0, "<Point %.1f %.1f>", x, y);
X x = (float) p1->i.x / 5.0f;
X y = (float) (mifstat.size.y - p1->i.y) / 5.0f;
X pout(&mifstat.pout, 0, "<Point %.1f %.1f>", x, y);
X mifstat.ga.in_line = TRUE;
X }
X/*
X * Store p1 of this line to see if it connects to next one
X */
X   p1last.i.x = p1->i.x;
X   p1last.i.y = p1->i.y;
X}
X
Xstatic int
Xmif_text(GR_PT *p, ATTR_STRUCT *a, TEXT_STRUCT *ta, char *text)
X{
X   float x,y,size;
X   int fsize;
X   int fnum;
X   int entry_font;
X   int swidth;
X   BOOL in_str;
X   BOOL sub_sup;
X   char *c,lastc;
X
X   PREPARE;
X   END_LAST_LINE; /* End previous line */
X   swidth = 0;
X   pout(&mifstat.pout, POUT_SL, "<TextLine");
X   if (a->color != mifstat.ga.color) { /* Color change */
X mifstat.ga.color = a->color;
X if (a->color == mifstat.ca.black) /* Spot Color */
X   pout(&mifstat.pout, 0, "<ObColor `Black'>");
X else if (a->color == mifstat.ca.white)
X   pout(&mifstat.pout, 0, "<ObColor `White'>");
X else if (a->color == mifstat.ca.red)
X   pout(&mifstat.pout, 0, "<ObColor `Red'>");

```

```

X else if (a->color == mifstat.ca.green)
X   pout(&mifstat.pout, 0, "<ObColor `Green'>");
X else if (a->color == mifstat.ca.blue)
X   pout(&mifstat.pout, 0, "<ObColor `Blue'>");
X else if (a->color == mifstat.ca.cyan)
X   pout(&mifstat.pout, 0, "<ObColor `Cyan'>");
X else if (a->color == mifstat.ca.magenta)
X   pout(&mifstat.pout, 0, "<ObColor `Magenta'>");
X else if (a->color == mifstat.ca.yellow)
X   pout(&mifstat.pout, 0, "<ObColor `Yellow'>");
X else {
X   printf("Line color cannot be spot color, using black\n");
X   pout(&mifstat.pout, 0, "<ObColor `Black'>");
X   mifstat.ga.color = mifstat.ca.black;
X }
X }
X if (ta->orien != 0.0f) /* Rotated text */
X pout(&mifstat.pout, 0, "<Angle %f>", ta->orien);
X/*
X * Text alignment must be set every time (default left)
X */
X if (ta->align != 0.0f) {
X if (ta->align == 0.5f)
X   pout(&mifstat.pout, 0, "<TLAlignment Center>");
X else
X   pout(&mifstat.pout, 0, "<TLAlignment Right>");
X }
X x = (float) p->i.x / 5.0f;
X y = (float) (mifstat.size.y - p->i.y) / 5.0f;
X/*
X * Correct for bias of y-axis text...too close to ticks
X */
X size = (ta->size == 0.0f) ? 1.0f : ta->size;
X fsize = (int) (size * mifstat.ta.size);
X if (ta->orien == 90.0f)
X x -= 1.5 * fsize;
X pout(&mifstat.pout, 0, "<TLOrigin %.1f %.1f>", x, y);
X/*
X * Font handling
X */
X if (mifstat.ta.newfont) { /* Font change */
X mif_font();
X mifstat.ta.newfont = FALSE;
X }
X if (fsize != mifstat.ta.cursize) { /* Size change */
X mifstat.ta.cursize = fsize;
X pout(&mifstat.pout, 0, "<Font <FSize %d>>", fsize);
X }

```

```

X/*
X * Now start processing the string; check for !-escapes. !c is handled
X * by IDL itself
X */
X  entry_font = mifstat.ta.psfont;
X  c = text;
X  in_str = FALSE;
X  sub_sup = FALSE;
X  while (*c != '\0') {
X if (*c == '!') { /* IDL Escape sequence */
X  switch (*(++c)) {
X   case '!': /* Just print ! */
X do_char(*c, &in_str, &swidth, sub_sup);
X break;
X   case 'B': /* Bullet Symbol */
X   case 'b':
X end_str(&in_str);
X pout(&mifstat.pout, 0, "<Char Bullet>");
X break;
X   case 'D': /* Subscript */
X   case 'd':
X end_str(&in_str);
X pout(&mifstat.pout, 0, "<Font <FPosition FSubscript>>");
X sub_sup = TRUE;
X break;
X   case 'M': /* Symbol (math) font */
X   case 'm':
X end_str(&in_str);
X mifstat.ta.psfont = 30;
X mif_font();
X break;
X   case 'N': /* Back to normal */
X   case 'n':
X end_str(&in_str);
X pout(&mifstat.pout, 0, "<Font <FPosition FNormal>>");
X sub_sup = FALSE;
X break;
X   case 'U': /* Superscript */
X   case 'u':
X end_str(&in_str);
X pout(&mifstat.pout, 0, "<Font <FPosition FSuperscript>>");
X sub_sup = TRUE;
X break;
X   case 'X': /* Return to entry font */
X   case 'x':
X end_str(&in_str);
X mifstat.ta.psfont = entry_font;
X mif_font();

```

```

X break;
X case '1': /* Font number in teens */
X fnum = 10 + *(++c) - '0'; /* Parse second digit */
X if (fnum < 10 || fnum > 19) { /* Bad digit */
X printf("Bad font specified\n");
X } else {
X end_str(&in_str);
X mifstat.ta.psfont = miffonts[fnum-3].ps_font_num;
X mif_font();
X }
X break;
X case '2': /* Font number twenty */
X fnum = 20;
X if (*(++c) != '0') {
X printf("Bad font specified\n");
X } else {
X end_str(&in_str);
X mifstat.ta.psfont = -1;
X mif_font();
X }
X break;
X case '3': /* Single-digit font */
X case '4':
X case '5':
X case '6':
X case '7':
X case '8':
X case '9':
X fnum = *c - '3';
X end_str(&in_str);
X mifstat.ta.psfont = miffonts[fnum].ps_font_num;
X mif_font();
X break;
X default:
X printf("Illegal escape sequence\n");
X break;
X }
X } else {
X do_char(*c, &in_str, &swidth, sub_sup); /* Send to do_char */
X }
X c++;
X }
X end_str(&in_str); /* End the string */
X lastc = *(c-1);
X swidth -= (int) ((sub_sup ? 0.8f : 1.0f) *
X (mifstat.ta.wx[lastc] - mifstat.ta.urx[lastc]));
X if (sub_sup) { /* Super- or sub-scripting */
X pout(&mifstat.pout, 0, "<Font <FPosition FNormal>>");

```

```

X sub_sup = FALSE;
X  }
X/*
X * Close bracket to end the line
X */
X  pout(&mifstat.pout, POUT_FL, ">");
X
X  return (int) (swidth * fsize / 200.0f);
X}
X
Xstatic void
Xmif_erase(ATTR_STRUCT *a)
X{
X
X  if (mifstat.state.not_blank) {
X/*
X * End the page.
X */
X END_LAST_LINE;  /* End previous line */
X  pout(&mifstat.pout, POUT_SL|POUT_FL, ">\tt# End of Page %d",
X    mifstat.state.cur_page);
X
X  mifstat.state.cur_page++;
X  }
X/*
X * Reset flag
X */
X  mifstat.state.not_blank = FALSE;
X}
X
X/*
X * POLYFILL routine
X */
Xstatic void
Xmif_poly(int xi[], int yi[], int n, POLYFILL_ATTR *pa)
X{
X  float x,y;
X  int i,color;
X
X  PREPARE;
X  END_LAST_LINE;  /* End previous line */
X  pout(&mifstat.pout, POUT_SL, "<Polygon");
X/*
X * IDL POLYFILL's are done with no border
X */
X  pout(&mifstat.pout, 0, "<Pen 15>");
X  mifstat.ga.pen = 15;
X  color = pa->attr->color;

```

```

X  if (color != mifstat.ga.color) { /* Color change */
X  mifstat.ga.color = color;
X  if (color == mifstat.ca.black) /* Spot Color */
X    pout(&mifstat.pout, 0, "<ObColor `Black'>");
X  else if (color == mifstat.ca.white)
X    pout(&mifstat.pout, 0, "<ObColor `White'>");
X  else if (color == mifstat.ca.red)
X    pout(&mifstat.pout, 0, "<ObColor `Red'>");
X  else if (color == mifstat.ca.green)
X    pout(&mifstat.pout, 0, "<ObColor `Green'>");
X  else if (color == mifstat.ca.blue)
X    pout(&mifstat.pout, 0, "<ObColor `Blue'>");
X  else if (color == mifstat.ca.cyan)
X    pout(&mifstat.pout, 0, "<ObColor `Cyan'>");
X  else if (color == mifstat.ca.magenta)
X    pout(&mifstat.pout, 0, "<ObColor `Magenta'>");
X  else if (color == mifstat.ca.yellow)
X    pout(&mifstat.pout, 0, "<ObColor `Yellow'>");
X  else {
X    printf("Line color cannot be spot color, using black\n");
X    pout(&mifstat.pout, 0, "<ObColor `Black'>");
X    mifstat.ga.color = mifstat.ca.black;
X  }
X  }
X  if (pa->extra.fill_style != mifstat.ga.fill) {
X  mifstat.ga.fill = pa->extra.fill_style;
X  pout(&mifstat.pout, 0, "<Fill %d>", mifstat.ga.fill);
X  }
X/*
X * Now place the points
X */
X  for (i=0; i<n; i++) {
X  x = (float) xi[i] / 5.0f;
X  y = (float) (mifstat.size.y - yi[i]) / 5.0f;
X  pout(&mifstat.pout, 0, "<Point %.1f %.1f>", x, y);
X  }
X/*
X * Close bracket to end the polygon
X */
X  pout(&mifstat.pout, POUT_FL, ">");
X}
X
Xstatic void
Xmif_color(long start, long n)
X{
X  int i;
X  UCHAR *red,*green,*blue;
X

```

```

X/*
X * Store arguments in mifstat
X */
X  mifstat.ca.start = start;
X  mifstat.ca.n = n;
X/*
X * Scan through table to find any spot colors
X */
X  red = (UCHAR *) color_map + start;
X  green = (UCHAR *) (red + COLOR_MAP_SIZE);
X  blue = (UCHAR *) (green + COLOR_MAP_SIZE);
X  for (i=0; i<n; i++,red++,green++,blue++) {
X if ((*red == 0) && (*green == 0) && (*blue == 0)) { /* Black */
X   mifstat.ca.black = i;
X   continue;
X }
X if ((*red == 255) && (*green == 255) && (*blue == 255)) { /* White */
X   mifstat.ca.white = i;
X   continue;
X }
X if ((*red == 255) && (*green == 0) && (*blue == 0)) { /* Red */
X   mifstat.ca.red = i;
X   continue;
X }
X if ((*red == 0) && (*green == 255) && (*blue == 0)) { /* Green */
X   mifstat.ca.green = i;
X   continue;
X }
X if ((*red == 0) && (*green == 0) && (*blue == 255)) { /* Blue */
X   mifstat.ca.blue = i;
X   continue;
X }
X if ((*red == 0) && (*green == 255) && (*blue == 255)) { /* Cyan */
X   mifstat.ca.cyan = i;
X   continue;
X }
X if ((*red == 255) && (*green == 0) && (*blue == 255)) { /* Magenta */
X   mifstat.ca.magenta = i;
X   continue;
X }
X if ((*red == 255) && (*green == 255) && (*blue == 0)) { /* Yellow */
X   mifstat.ca.yellow = i;
X   continue;
X }
X }
X}
X
X/*

```

```

X * Create FramedImage from TV data.  Uses 8-bit Pseudocolor.
X */
Xstatic void
Xmif_image(UCHAR *data, int x0, int y0, int nx, int ny, int dir,
X TV_STRUCT *secondary)
X{
X  float l,t,w,h; /* Left, top, width, height of image */
X  RasterfileT rdat; /* FramedImage Raster header */
X  int nbytes;
X
X  PREPARE;
X  END_LAST_LINE; /* End previous line */
X#ifdef DEBUG
X  printf("Bitmap image (%dx%d) being created\n", nx, ny);
X#endif
X  pout(&mifstat.pout, POUT_SL, "<ImportObject");
X/*
X * IDL TV's are done with no border
X */
X  pout(&mifstat.pout, 0, "<Pen 15>");
X  mifstat.ga.pen = 15;
X  pout(&mifstat.pout, 0, "<ImportObFile `2.0 internal inset'>");
X/*
X * Calculate size of bitmap.  Using scalable pixels
X */
X  l = (float) x0 / 5.0f;
X  t = (float) (mifstat.size.y - y0 - secondary->ysize) / 5.0f;
X  w = (float) secondary->xsize / 5.0f;
X  h = (float) secondary->ysize / 5.0f;
X  pout(&mifstat.pout, 0, "<ShapeRect %.1f %.1f %.1f %.1f>", l, t, w, h);
X  pout(&mifstat.pout, 0, "<ImportObFixedSize Yes>");
X/*
X * Set rotation and left-right flip to get bitmap orientation right
X * This works for default order; I'm guessing for other.
X */
X  if (secondary->order) {
X#ifdef DEBUG
X  printf("Bitmap image order true, not flipping\n");
X#endif
X  } else {
X#ifdef DEBUG
X  printf("Bitmap image order false, flipping\n");
X#endif
X  pout(&mifstat.pout, 0, "<Angle 180>");
X  pout(&mifstat.pout, 0, "<FlipLR Yes>");
X  }
X/*
X * Now print out the image.  Set up mifstat.pout for leading ampersands

```



```

X */
X  pout(&mifstat.pout, POUT_SL|POUT_FL, "=FrameImage");
X  pout(&mifstat.pout, POUT_SL, "&%%v");
X  pout(&mifstat.pout, POUT_SL, "&\\x");
X  mifstat.pout.leading = "&";
X  mifstat.pout.leading_len = 1;
X/*
X * Create FrameImage header
X */
X  nbytes = nx*ny;
X  rdat.ras_magic = RAS_MAGIC;
X  rdat.ras_width = nx;
X  rdat.ras_height = ny;
X  rdat.ras_depth = 8;
X  rdat.ras_length = 8*nbytes; /* Ignored */
X  rdat.ras_type = RT_STANDARD; /* No RLE (yet?) */
X  rdat.ras_maptype = RMT_EQUAL_RGB;
X  rdat.ras_maplength = 3*COLOR_MAP_SIZE; /* Must be 3*256 */
X  write_hex((UCHAR *) &rdat, sizeof(RasterfileT));
X/*
X * Write out color map. IDL color map is already in the correct format
X */
X  write_hex((UCHAR *)(color_map + mifstat.ca.start), 3*COLOR_MAP_SIZE);
X/*
X * Then the image bytes; each line must be done at a time, because if
X * it has an odd number of pixels, it must be padded to a word
X * boundary
X */
X  for (iy = 0; iy < ny; iy++, data+=nx) {
X  write_hex(data, nx);
X  if (nx % 2)
X    IDL_Pout(&mifstat.pout, IDL_POUT_NOSP|IDL_POUT_LEADING, "00");
X  }
X/*
X * End the inset data.
X */
X  pout(&mifstat.pout, POUT_NOSP|POUT_LEADING, "\\x");
X  mifstat.pout.leading = NULL;
X  mifstat.pout.leading_len = 0;
X  pout(&mifstat.pout, POUT_SL|POUT_FL, "=EndInset");
X/*
X * Close bracket to end the image
X */
X  pout(&mifstat.pout, POUT_FL, ">");
X}
X
Xstatic void
Xmif_device(int argc, VPTR *argv, char *argk)

```

```

X{
X  static long close_file; /* CLOSE_FILE keyword */
X  static VPTR funit_argv[] = {&mifstat.state.lun};
X  static int fname_p; /* TRUE if FILENAME specified */
X  static STRING fname; /* FILENAME */
X  static long units; /* Units for x,y */
X  static int xsize_p; /* Non-zero if XSIZE */
X  static int ysize_p; /* Non-zero if YSIZE */
X  static float xsize; /* Width */
X  static float ysize; /* Height */
X  static long ffamily; /* Font family */
X  static long fweight; /* Font weight */
X  static long fangle; /* Font angle */
X  static long fvariation; /* Font variation */
X  static int findex_p; /* TRUE if FONT_INDEX keyword */
X  static long findex; /* Font index to map to */
X  static int fsize_p; /* TRUE if FONT_SIZE */
X  static long fsize; /* Hardware font size (in pt) */
X  static int ufont_p; /* TRUE if USER_FONT keyword */
X  static STRING ufont; /* USER_FONT name */
X
X/*
X * Defines for units used in X,Y size, offset keywords
X */
X#define UNIT_CM 0 /* CM (default) */
X#define UNIT_PT 1 /* Points */
X#define UNIT_IN 2 /* Inches */
X
X/*
X * Array defining keyword parameters; must be in lexical order!
X */
X  static KW_PAR kw_list[] = {
X  KW_FAST_SCAN, /* Might speed things up */
X  { "AVANTGARDE", TYP_LONG, 1,
X    KW_VALUE|(1<<2), NULL, CHARA(ffamily) },
X  { "BKMAN", TYP_LONG, 1,
X    KW_VALUE|(1<<3), NULL, CHARA(ffamily) },
X  { "BOLD", TYP_LONG, 1,
X    KW_VALUE|(1<<1), NULL, CHARA(fweight) },
X  { "BOOK", TYP_LONG, 1,
X    KW_VALUE|(1<<3), NULL, CHARA(fweight) },
X  { "CLOSE_FILE", TYP_LONG, 1,
X    KW_ZERO, NULL, CHARA(close_file) },
X  { "COURIER", TYP_LONG, 1,
X    KW_VALUE|(1<<0), NULL, CHARA(ffamily) },
X  { "DEMI", TYP_LONG, 1,
X    KW_VALUE|(1<<4), NULL, CHARA(fweight) },
X  { "FILENAME", TYP_STRING, 1,

```

```

X    NULL, &fname_p, CHARA(fname) },
X { "FONT_INDEX", TYP_LONG, 1,
X    NULL, &findex_p, CHARA(findex) },
X { "FONT_SIZE", TYP_LONG, 1,
X    NULL, &fsize_p, CHARA(fsize) },
X { "HELVETICA", TYP_LONG, 1,
X    KW_VALUE|(1<<1), NULL, CHARA(ffamily) },
X { "INCHES", TYP_LONG, 1,
X    KW_VALUE|UNIT_IN, NULL, CHARA(units) },
X { "ITALIC", TYP_LONG, 1,
X    KW_VALUE|(1<<1), NULL, CHARA(fangle) },
X { "LIGHT", TYP_LONG, 1,
X    KW_VALUE|(1<<2), NULL, CHARA(fweight) },
X { "MEDIUM", TYP_LONG, 1,
X    KW_VALUE|(1<<0), NULL, CHARA(fweight) },
X { "NARROW", TYP_LONG, 1,
X    KW_VALUE|(1<<1), NULL, CHARA(fvariation) },
X { "OBLIQUE", TYP_LONG, 1,
X    KW_VALUE|(1<<2), NULL, CHARA(fangle) },
X { "PALATINO", TYP_LONG, 1,
X    KW_VALUE|(1<<7), NULL, CHARA(ffamily) },
X { "POINTS", TYP_LONG, 1,
X    KW_VALUE|UNIT_PT, NULL, CHARA(units) },
X { "SCHOOLBOOK", TYP_LONG, 1,
X    KW_VALUE|(1<<6), NULL, CHARA(ffamily) },
X { "SYMBOL", TYP_LONG, 1,
X    KW_VALUE|(1<<8), NULL, CHARA(ffamily) },
X { "TIMES", TYP_LONG, 1,
X    KW_VALUE|(1<<9), NULL, CHARA(ffamily) },
X { "USER_FONT", TYP_STRING, 1,
X    NULL, &ufont_p, CHARA(ufont) },
X { "XSIZE", TYP_FLOAT, 1,
X    NULL, &xsize_p, CHARA(xsize) },
X { "YSIZE", TYP_FLOAT, 1,
X    NULL, &ysize_p, CHARA(ysize) },
X { "ZAPFCHANCERY", TYP_LONG, 1,
X    KW_VALUE|(1<<4), NULL, CHARA(ffamily) },
X { "ZAPFDINGBATS", TYP_LONG, 1,
X    KW_VALUE|(1<<5), NULL, CHARA(ffamily) },
X { NULL }
X };
X/*
X * Local variables
X */
X PS_FONT_T new_font; /* Description of new PS font */
X int i;
X float sfactor; /* Convert size, offset to pixels */
X

```

```

X/*
X * Keyword processing. Must zero flag keywords before processing
X */
X  ffamily = fweight = fangle = fvariation = 0;
X  units = 0;
X  (void) get_kw_params(argc, argv, argk, kw_list, (VPTR *) NULL, 1);
X/*
X * CLOSE_FILE keyword
X */
X  if (close_file && mifstat.state.fopen) {
X  mif_closefile();
X  free_lun(1, funit_argv);
X  }
X/*
X * FILENAME keyword
X */
X  if (fname_p) {
X  (void) strncpy(mifstat.state.fname, STRING_STR(&fname),
X      MAX_PATH_LEN-1);
X  mif_openfile();
X  }
X/*
X * FONT_SIZE keyword
X */
X  if (fsize_p) {
X  mifstat.ta.size = fsize;
X  mif_dev.ch_size[0] = fsize * DEF_XCSIZE / 12;
X  mif_dev.ch_size[1] = fsize * DEF_YCSIZE / 12;
X  }
X/*
X * A font was specified. Inherit previous characteristics UNLESS a new
X * font family was specified; then set to MEDIUM REGULAR NORMAL before
X * checking other keywords.
X */
X  if (ffamily || fweight || fangle || fvariation) {
X  new_font = psfonts[mifstat.ta.psfont];
X  if (ffamily) {
X    new_font.family = (TEXT_FFAM) find_bit(ffamily);
X    new_font.weight = MEDIUM;
X    new_font.angle = REGULAR;
X    new_font.variation = NORMAL;
X  }
X  if (fweight)
X    new_font.weight = (TEXT_FWGT) find_bit(fweight);
X  if (fangle)
X    new_font.angle = (TEXT_FANG) find_bit(fangle);
X  if (fvariation)
X    new_font.variation = (TEXT_FVAR) find_bit(fvariation);

```

```

X#ifdef DEBUG
X printf("New font requested:\n\tFamily: %s\n\tWeight: %s\n\tAngle: %s\n\tVariation: %s\n",
X     family_names[new_font.family], weight_names[new_font.weight],
X     angle_names[new_font.angle],
X     variation_names[new_font.variation]);
X#endif
X for (i=0; i < N_PS_FONTS; i++)
X     if ((new_font.family == psfonts[i].family) &&
X (new_font.weight == psfonts[i].weight) &&
X (new_font.angle == psfonts[i].angle) &&
X (new_font.variation == psfonts[i].variation)) {
X/*
X * Got a match, now check for FONT_INDEX keyword
X */
X#ifdef DEBUG
X printf("Match found, PS font %d\n", i);
X#endif
X if (findindex_p) {
X     findindex -= 3;
X     miffonts[findindex].ps_font_num = i;
X     (void) strcpy(miffonts[findindex].name, psfonts[i].name);
X } else {
X     mifstat.ta.psfont = i;
X     mifstat.ta.newfont = TRUE;
X }
X break;
X }
X if (i == N_PS_FONTS) /* Print error if no match */
X     printf("Font not available, using %s\n",
X     psfonts[mifstat.ta.psfont].name);
X }
X/*
X * USER_FONT keyword
X */
X if (ufont_p) {
X mifstat.ta.newfont = TRUE;
X mifstat.ta.psfont = -1;
X     (void) strncpy(miffonts[17].name, STRING_STR(&ufont),
X     MAX_FONT_NAME_LEN-1);
X }
X/*
X * XSIZE, YSIZE keywords. First handle units, then calculate #'s, then
X * recompute aspect ratio. Size can only be changed on the first page
X */
X if (xsize_p || ysize_p) {
X if (mifstat.state.not_blank || (mifstat.state.cur_page > 1)) {
X     printf("Page already sized; close this file and try again\n");
X } else {

```

```

X  switch (units) {
X  case UNIT_CM:
X  sfactor = (float) PIXELS_INCH / 2.54f;
X  break;
X  case UNIT_PT:
X  sfactor = 5.0f;
X  break;
X  case UNIT_IN:
X  sfactor = (float) PIXELS_INCH;
X  break;
X  }
X  if (xsize_p)
X  mif_dev.t_size[0] = mif_dev.v_size[0] = mifstat.size.x =
X  (int) (sfactor * xsize + 0.5f);
X  if (ysize_p)
X  mif_dev.t_size[1] = mif_dev.v_size[1] = mifstat.size.y =
X  (int) (sfactor * ysize + 0.5f);
X  mif_dev.aspect = (float) mif_dev.v_size[0] /
X  (float) mif_dev.v_size[1];
X/*
X * Calculate size info in points
X */
X  mifstat.size.width = (float) mifstat.size.x / 5.0f;
X  mifstat.size.height = (float) mifstat.size.y / 5.0f;
X }
X }
X/*
X * Clean up and exit
X */
X#undef UNIT_CM
X#undef UNIT_PT
X#undef UNIT_IN
X  keyword_cleanup(KW_CLEAN_ALL);
X}
X
Xstatic void
Xmif_help(int argc, VPTR *argv)
X{
X  int i;
X  char str[8];
X
X#ifdef DEBUG
X  printf("MIF Driver Version: %s\n", mif_version);
X#endif
X  printf("  File: %s\n", (mifstat.state.fopen ? mifstat.state.fname :
X  "<none>"));
X  printf("  Size (X,Y): (%.2f,%.2f) cm, (%.2f,%.2f) in, (%.1f,%.1f) pt\n",
X  mifstat.size.width * 2.54f / 72.0f,

```

```

X  mifstat.size.height * 2.54f / 72.0f,
X  mifstat.size.width / 72.0f, mifstat.size.height / 72.0f,
X  mifstat.size.width, mifstat.size.height);
X  printf("  Font Size: %d\n", mifstat.ta.size);
X  printf("  Font: %s\n", psfonts[mifstat.ta.psfont].name);
X  printf("  Font Mapping:\n");
X  for (i=0; i < NUM_FONTS; i++) {
X  sprintf(str, "(!%d)", i+3);
X  printf("%12s %-30s", str, miffonts[i].name);
X  if (++i < NUM_FONTS) {
X    sprintf(str, "(!%d)", i+3);
X    printf("%5s %-30s\n", str, miffonts[i].name);
X  } else
X    printf("\n");
X  }
X}
X
Xstatic void
Xmif_openfile(void)
X{
X  static char *filename = mifstat.state.fname;
X  VPTR argv[2];
X
X/*
X * If a file is already open, close it and swipe its LUN.  Otherwise
X * must use get_lun.
X */
X  argv[0] = &mifstat.state.lun;
X  if (mifstat.state.fopen)
X  mif_closefile();
X  else {
X  get_lun(1, argv);
X  declare_exit_handler(mif_closefile); /* Register exit handler */
X  mif_dev.unit = mifstat.state.lun.value.l;
X  mifstat.pout.unit = mifstat.state.lun.value.l;
X  }
X/*
X * Open the file - argv[0] is LUN from above
X */
X  argv[1] = ret_str_as_STRING(filename);
X  open_file(2, argv, (char *) NULL, OPEN_W|OPEN_NEW, F_NOCLOSE);
X  mifstat.state.fopen = TRUE;
X  mifstat.state.not_blank = FALSE;
X  mifstat.state.cur_page = 1;
X  mifstat.pout.curcol = 0;
X#ifdef DEBUG
X  printf("File %s opened, LUN %d\n", filename, mif_dev.unit);
X#endif

```

```

X/*
X * Delete temporary variable created by ret_str_as_STRING
X */
X  DELTMP(argv[1]);
X}
X
Xstatic void
Xmif_prefile(void)
X{
X
X/*
X * Check if file is open.  If not, (re)use current file name
X */
X  if (!mifstat.state.fopen)
X  mif_openfile();
X/*
X * If it is the first page, write document header.  By default, we will
X * create a page of specified size and no margins
X */
X  if (mifstat.state.cur_page == 1) {
X  pout(&mifstat.pout, POUT_SL|POUT_FL,
X    "<MIFFile 4.00>\t# Generated by IDL MIF driver Version %s",
X    mif_version);
X  pout(&mifstat.pout, POUT_FL, "<Units Upt>");
X  pout(&mifstat.pout, POUT_SL, "<Document>");
X  pout(&mifstat.pout, 0, "<DPageSize %.1f %.1f>",
X    mifstat.size.width,mifstat.size.height);
X  pout(&mifstat.pout, POUT_FL, ">");
X  }
X/*
X * Start the page
X */
X  pout(&mifstat.pout, POUT_FL, "<Page\t\t# Start of page %d",
X  mifstat.state.cur_page);
X
X  mifstat.state.not_blank = TRUE;
X}
X
Xstatic void
Xmif_closefile(void)
X{
X  static VPTR argv[] = {&mifstat.state.lun};
X
X  if (mifstat.state.not_blank) {
X/*
X * End the page.
X */
X  END_LAST_LINE;  /* End previous line */

```



```

X pout(&mifstat.pout, POUT_SL|POUT_FL, ">\t\t# End of Page %d",
X   mifstat.state.cur_page);
X mifstat.state.not_blank = FALSE;
X }
X if (mifstat.state.fopen) {
X/*
X * Remove restriction on closing file, then close it. Also, reset
X * file name to default idl.mif
X */
X FILE_CLOSE(mifstat.state.lun.value.l);
X close_file(1, argv, (char *) NULL);
X mifstat.state.fopen = FALSE;
X strcpy(mifstat.state.fname, "idl.mif");
X mif_dev.unit = 0;
X }
X/*
X * Reset some variables to their defaults
X */
X   mifstat.state.cur_page = 1;
X   mifstat.ta.newfont = TRUE;
X   mifstat.ta.cursize = 0;
X   mifstat.ga.color = mifstat.ca.black;
X   mifstat.ga.thick = -1.0f;
X   mifstat.ga.linestyle = -1;
X   mifstat.ga.pen = 0;
X   mifstat.ga.fill = -1;
X}
X
X/*
X * Write out MIF statements for IDL linestyles
X */
Xstatic void
Xmif_linestyle(int linestyle)
X{
X
X   pout(&mifstat.pout, 0, "<DashedPattern <DashedStyle");
X   switch (linestyle) {
X   case 0: /* Solid line */
X   pout(&mifstat.pout, 0, "Solid>");
X   break;
X   case 1: /* Dotted */
X   pout(&mifstat.pout, 0, "Dashed>");
X   pout(&mifstat.pout, 0, "<DashSegment 2>");
X   pout(&mifstat.pout, 0, "<DashSegment 4>");
X   break;
X   case 2: /* Dashed */
X   pout(&mifstat.pout, 0, "Dashed>");
X   pout(&mifstat.pout, 0, "<DashSegment 8>");

```

```

X pout(&mifstat.pout, 0, "<DashSegment 6>");
X break;
X case 3: /* Dash Dot */
X pout(&mifstat.pout, 0, "Dashed>");
X pout(&mifstat.pout, 0, "<DashSegment 12>");
X pout(&mifstat.pout, 0, "<DashSegment 6>");
X pout(&mifstat.pout, 0, "<DashSegment 2>");
X pout(&mifstat.pout, 0, "<DashSegment 6>");
X break;
X case 4: /* Dash Dot Dot Dot */
X pout(&mifstat.pout, 0, "Dashed>");
X pout(&mifstat.pout, 0, "<DashSegment 12>");
X pout(&mifstat.pout, 0, "<DashSegment 6>");
X pout(&mifstat.pout, 0, "<DashSegment 2>");
X pout(&mifstat.pout, 0, "<DashSegment 6>");
X pout(&mifstat.pout, 0, "<DashSegment 2>");
X pout(&mifstat.pout, 0, "<DashSegment 6>");
X pout(&mifstat.pout, 0, "<DashSegment 2>");
X pout(&mifstat.pout, 0, "<DashSegment 6>");
X break;
X case 5: /* Long Dashes */
X pout(&mifstat.pout, 0, "Dashed>");
X pout(&mifstat.pout, 0, "<DashSegment 16>");
X pout(&mifstat.pout, 0, "<DashSegment 10>");
X break;
X }
X pout(&mifstat.pout, 0, ">");
X}
X
X/*
X * Print out new font definition
X */
Xstatic void
Xmif_font(void)
X{
X int font;
X short i,cnum,wx,llx,urx;
X char afmfile[512];
X char instr[256],cname[32];
X FILE *fp;
X VARIABLE lun;
X VPTR argv[2];
X FILE_STAT fstat;
X
X font = mifstat.ta.psfont;
X/*
X * If it is a standard font, use family, etc.; otherwise, use the name
X * provided

```

```

X */
X  if (font != -1) {
X  pout(&mifstat.pout, 0, "<Font");
X  pout(&mifstat.pout, 0, "<FFamily '%s'",
X    family_names[psfonts[font].family]);
X  pout(&mifstat.pout, 0, "<FAngle '%s'",
X    angle_names[psfonts[font].angle]);
X  pout(&mifstat.pout, 0, "<FWeight '%s'",
X    weight_names[psfonts[font].weight]);
X  pout(&mifstat.pout, 0, "<FVar '%s'>>",
X    variation_names[psfonts[font].variation]);
X#ifdef DEBUG
X  printf("Font '%s' selected\n", psfonts[font].name);
X#endif
X/*
X * If it is a standard font, then retrieve encoding information
X */
X (void) sprintf(afmfile, "%s/finit/fontdir/%s.afm",
X   getenv("FMHOME"), psfonts[font].afm_name);
X argv[0] = &lun;
X lun.type = TYP_LONG;
X lun.flags = 0;
X get_lun(1, argv);
X argv[1] = ret_str_as_STRING(afmfile);
X open_file(2, argv, (char *) NULL, OPEN_R, F_NOCLOSE);
X get_file_stat(lun.value.l, &fstat);
X if ((fp = fstat.fptr) == (FILE *) NULL) {
X   printf("Maker font metric file not found, assuming fixed-width\n");
X   for (i=0; i<256; i++) {
X     mifstat.ta.wx[i] = 600;
X     mifstat.ta.llx[i] = 0;
X     mifstat.ta.urx[i] = 600;
X   }
X } else {
X   if (font == 30 || font == 21) { /* Symbol or Zapf Dingbats */
X     while(fgets(instr, 256, fp) != (char *) NULL) {
X       if (sscanf(instr,
X         "C %hd ; WX %hd ; N %s ; B %hd %d %hd %d ;",
X         &cnum, &wx, &llx, &urx) == 4
X         && cnum > 0 && cnum < 256) {
X         mifstat.ta.wx[cnum] = wx;
X         mifstat.ta.llx[cnum] = llx;
X         mifstat.ta.urx[cnum] = urx;
X       }
X     }
X   } else { /* Regular font */
X     while(fgets(instr, 256, fp) != (char *) NULL) {
X       if (sscanf(instr,

```

```

X      "C %*d ; WX %hd ; N %s ; B %hd %*d %hd %*d ;",
X      &wx, cname, &llx, &urx) == 4) {
X  for (i=0; i<256; i++)
X    if (strcmp(cname, Encoding[i]) == 0) {
X      mifstat.ta.wx[i] = wx;
X      mifstat.ta.llx[i] = llx;
X      mifstat.ta.urx[i] = urx;
X    }
X  }
X }
X }
X }
X FILE_CLOSE(lun.value.l);
X free_lun(1, argv);
X } else {
X  pout(&mifstat.pout, 0, "<Font <FPostScriptName `%'>>",
X    miffonts[17].name);
X/*
X * For a user font, assume fixed-width
X */
X for (i=0; i<256; i++) {
X   mifstat.ta.wx[i] = 600;
X   mifstat.ta.llx[i] = 0;
X   mifstat.ta.urx[i] = 600;
X }
X#ifdef DEBUG
X printf("User Font '%s' selected\n", miffonts[17].name);
X#endif
X }
X  deltmp(argv[1]); /* Clean up temporary variable */
X#ifdef DEBUG
X   for (i=0; i<256; i++)
X   printf("%8d", mifstat.ta.wx[i]);
X   printf("\n");
X   for (i=0; i<256; i++)
X   printf("%8d", mifstat.ta.llx[i]);
X   printf("\n");
X   for (i=0; i<256; i++)
X   printf("%8d", mifstat.ta.urx[i]);
X   printf("\n");
X#endif
X}
X/*
X * Routines for outputting, ending text strings
X */
Xstatic void
Xdo_char(UCHAR c, BOOL *in, int *width, BOOL sub_sup)
X{

```

```

X float cfac;
X
X if (!*in) { /* Start new string */
X *in = TRUE;
X pout(&mifstat.pout, POUT_SL, "<String `");
X }
X if (c > 0x7f) /* 8-bit ASCII */
X pout(&mifstat.pout, POUT_NOSP|POUT_NOBREAK, "\\x%x ");
X else if (c == '\t') /* Tab character */
X pout(&mifstat.pout, POUT_NOSP|POUT_NOBREAK, "\\t");
X else if (c == '>') /* > character */
X pout(&mifstat.pout, POUT_NOSP|POUT_NOBREAK, "\\>");
X else if (c == '"') /* Quote character */
X pout(&mifstat.pout, POUT_NOSP|POUT_NOBREAK, "\\q");
X else if (c == '`') /* Backquote character */
X pout(&mifstat.pout, POUT_NOSP|POUT_NOBREAK, "\\Q");
X else if (c == '\\') /* Backslash character */
X pout(&mifstat.pout, POUT_NOSP|POUT_NOBREAK, "\\\\");
X else /* Default */
X pout(&mifstat.pout, POUT_NOSP|POUT_NOBREAK, "%c", c);
X/*
X * Add character width (from font metrics) to string width
X */
X cfac = sub_sup ? 0.8f : 1.0f;
X if (*width == 0) *width = (int) (-cfac * mifstat.ta.llx[c]);
X *width += (int) (cfac * mifstat.ta.wx[c]);
X if (mifstat.ta.wx[c] == 0)
X printf("Warning: Character \\x%x not in Maker character set\n", c);
X}
X
Xstatic void
Xend_str(BOOL *in)
X{
X
X if (*in) { /* In a string; end it */
X pout(&mifstat.pout, POUT_NOSP|POUT_NOBREAK, ">");
X *in = FALSE;
X }
X}
X
X/*
X * Write out binary data in hex format
X */
Xstatic void
Xwrite_hex(UCHAR *data, int n)
X{
X int i;
X for (i=0; i<n; i++, data++)

```

```

X pout(&mifstat.pout, POUT_NOSP|POUT_LEADING, "%02X", *data);
X}
X
X/*
X * Find which bit has been set; warn if multiple ones. Use first one.
X */
Xstatic int
Xfind_bit(long in)
X{
X    int i;
X
X    for (i=0; i < 32; i++)
X if (in & (1 << i)) {
X     in -= (1 << i); /* Subtract out that bit */
X     break;
X }
X    if (in)
X printf("find_bit: Warning...conflicting bits set\n");
X
X    return i;
X}
END_OF_FILE
if test 46780 -ne `wc -c <'mif_3.6.c'`; then
    echo shar: \"mif_3.6.c\" unpacked with wrong size!
fi
# end of 'mif_3.6.c'
fi
echo shar: End of archive 1 \ (of 2\).
cp /dev/null ark1isdone
MISSING=""
for I in 1 2 ; do
    if test ! -f ark${I}isdone ; then
MISSING="${MISSING} ${I}"
    fi
done
if test "${MISSING}" = "" ; then
    echo You have unpacked both archives.
    rm -f ark[1-9]isdone
else
    echo You still need to unpack the following archives:
    echo "      " ${MISSING}
fi
## End of shell archive.
exit 0

```

-----7A83A0A1267--
