The ROCKS Routines
Plotting Interface (V2)
(C Implementation)
Programmer's Guide

by

George N. Reeke, Jr.
The Rockefeller University
New York, N. Y. 10021

April 24, 1998
Revised, May 2, 2006
Revised, October 10, 2010
Revised, March 9, 2019

INTRODUCTION

        This document describes the plotting interface defined for the C language imple-
mentation of the ROCKS library. The ROCKS plotting routines support generation of a
"graphics metafile" concurrently with X graphics. Metafile formats are defined in this
document. General information about the ROCKS routines that are not related to plotting
is contained in a separate file, APNDX5.PDF.

The plotting functions represent a considerable extension of the set of routines ori-
ginally described in the CALCOMP manual "Programming CALCOMP Electromechanical Plotters"
(1974). The ROCKS routines are designed to provide a "lowest common denominator" system
that can be easily implemented on most graphical displays and output devices. Multiple
drawing windows, cached subplots, and multiple movable frames within a window are now
supported. Different windows may have different lifetimes; the plotting within each
window is assigned a unique plot sequence number at the time it starts. The decimal
encoding of metafiles used in previous versions of the plot library is no longer
supported–existing metafiles can be converted to the new binary encoding with a utility
program for viewing. Three-dimensional plotting routines, lighting effects, and ray
tracing are not provided in the current version.

SUMMARY OF PLOTTING ROUTINES

        When in X graphics, a default graphics window is always open. Call **newwin()** to
open a new (secondary) graphics window. Call **newplt()** to begin each new plot in the
default window (window 0) or **newpltw()** to begin a new plot in a previously defined
secondary window. Call **chgwin()** to switch drawing to a previously defined window. Call
retrace() to modify the thickness of drawn lines and **gmode()** to set the drawing mode.
Call **plot()** to draw points, **line()** to draw lines, **polyln()** to draw multi-segment lines,
polygn() to draw closed polygons or stars, **rect()** to draw rectangles, and **square()** to
draw squares. Call **arc()** or **arc2()** to draw arcs, **arrow()** to draw arrows, **circle()** to
draw circles, **ellips()** to draw ellipses. Call **mfprtf()** to plot formatted text, or call
simbol() or **simbc()** to plot text alone and **nomber()** or **nombc()** to plot numbers as text.
(For compatibility with earlier versions of the library, **symbol()**, **symbc()**, **number()**,
and **numbc** are also provided.) Call **axlin()** to draw and label a linear graph axis,
axlog() to draw and label a logarithmic graph axis, and **axpol()** to draw and label axes
for a polar plot. Call **bitmap()** or **bitmaps()** to plot a monochrome, gray scale, or
colored pixel image. Call **gobjid()** to assign an id number to a graphical object for
mouse selection. Call **plot()** to change the origin and **factor()** to rescale a drawing,
gscale() to interrogate the current scaling factor, and **where()** to interrogate the
current pen location and scale factor. Call **updwin()** to indicate that the specified
window should be redrawn to reflect all plotting commands entered up to this point
(implicit in a **newpltw()** call). Call **finplt()** to indicate that the current drawing in
the default window is complete (implicit in a **newplt()** call). Call **clswin()** to destroy
a numbered window and **endplt()** to close all windows and clean up the plotter interface
at the end of execution. Call **plotvers()** to determine the subversion repository version
number of the current compilation of the plot library.

Multiple plotting frames (also called "viewports" in this document) within a window are
now supported. Frames may be draggable or not draggable, bordered or not bordered,
rotated or not rotated, dynamically updated or not, their plotting commands stored for
reuse or not stored. Use **frmdef()** to define a frame and switch drawing to that frame;
frame() to switch drawing to a previously defined frame; **frmplt()** to copy plot commands
from a stored frame to the current drawing window; **frmdbm()** to draw a frame in current
state from last bookmark and set new bookmark; **frminq()** to obtain the coordinate trans-
formation for a frame that a user may have dragged; **frmpush()** to copy the current frame
onto a push-down stack; **frmpop()** to return to the frame most recently pushed with
frmpush(); and **frmdel()** to delete codes for a stored frame.

Specifications of colors, fonts, and line (originally pen) types involve character
strings. The library supports two ways to specify these items: the routines **pencol()**,
font(), and **pentyp()** allow them to be specified by name at any time. Alternatively,
routines **regcol()**, **regfont()**, and **regpen()** allow the user to register names for later
use. An integer registration number is returned that can be used in later calls to
rcolor(), **rfont()**, and **rpen()**, respectively, to invoke the registered color, font, or

pen type by number. A few registration numbers are predefined by compile-time names in
the header plotdefs.h, q.v.; these can be used without registering them. These include
a few common color names (e.g. RED, GREEN, BLUE, etc.), fonts (e.g. SWISS, ROMAN, etc.),
and some pen types that actually control line drawing style (e.g. SOLID, DOT, DASH, DOT-
DASH, etc.). A maximum of 2^31 color names and 2^15 font and pen names may be register-
ed, minus the number of built-in registered names. Additionally, a call to **color()**
allows one to specify numeric red-green-blue values. [An obsolete use of **color()** was to
select a hard-copy drawing pen by number. If needed, this can now be accomplished by a
call to **rcolor()** using of one of the preregistered color numbers.]

Registered color, font, and pen type numbers may differ on different nodes of a parallel
computer, but registration numbers returned on Node 0 may be copied to other nodes where
their use will invoke the same item. Doing this is the way to avoid the overhead of
registering the same items on all nodes.

Function **setmf()** controls an interface for recording plots in a machine-independent
"metafile" for later playback or immediate display via X graphics. Function **setmovie()**
controls an X user interface to hold plots for viewing until a particular keyboard or
mouse key is pressed and to operate an external film recording device. These control
routines are described below--it should be understood that they may not be implemented
on all systems that implement the standard plotting routines.

Coordinate Systems

      It is sometimes necessary to generate hardcopy plots to some absolute scale,
whereas visual displays of the same plots should normally be scaled to the largest size
that will fit the current window. The ROCKS plotting routines support both situations.
Absolute-scale devices interpret all plotting coordinates as being in inches. Startup
function **setmf()** allows the user to specify both the range and precision of the coordi-
nates stored in a metafile (default 32" range and 10-bit fractions). Relative-scale
devices scale the coordinates such that the dimensions xsiz,ysiz given in the **newplt()**
or **newpltw()** calls will just fit in the available plotting area. Hence, for such
devices the plotting units are defined implicitly by these parameters.

X increases to the right and y increases up from the current origin. Plot objects
extending beyond the xsiz,ysiz limits are clipped where supported. Local plotting
frames may be defined by calling **frmdef()**. In some implementations, the user may be
able to use a pointing device (e.g. a mouse) to rescale and move frames individually.
An entire plot may be rescaled by calling **factor()** before any drawing calls. A local
origin within the current frame may be set by calling **plot()**. (These changes operate
only in the current drawing window.)

Angle arguments to plotting functions are in degrees.

Specifying Colors

      As explained above, three methods are available for specifying colors. Routines
newplt(), **newpltw()**, **pencol()**, and **penset()** specify colors explicitly as English names
(e.g. "RED") or as hexadecimal blue-green-red values (referred to as "rgb" colors
below). The implementation picks the closest available color to the one requested.
These routines can handle literal color names entered by users. However, they may be
slow because of the need to match the color specified in each call with the available
device colors. Routine **regcol()** allows this lookup to be performed only once. It
registers a color name and returns an integer that can be used with **rcolor()** to switch
quickly to that color. Finally, routine **color()** allows colors to be specified directly
as numerical red-green-blue values.

In order to understand how color control works in the ROCKS plotting library, the
following design features should be kept in mind:

      (1) All color setting routines specify a "current drawing color". Subsequent
plotting commands use that color until a new one is specified. Each drawing frame, and,
on parallel computers, each computational node, has its own current drawing color, which
is unaffected by color changes on other frames or nodes.

     (2) The number of bits used to represent colors in bitmaps can be specified
individually for each bitmap. Black-and-white bitmaps use 1 bit per pixel, grayscale
bitmaps 8 or 16 bits per pixel, and colored bitmaps either 8 (obsolete), 24, or 48 bits
per pixel. Other colors are specified by one of the methods outlined above.

     (3) When a metafile is rendered on a visual display device, a color lookup table
may be used to map metafile colors to the colors actually supported by the device. On
newer devices, 24- or 32-bit color is routinely available and mapping is not a concern.
With older devices, the hardware or operating system may or may not permit applications
to modify the lookup table. Therefore, the mapping of metafile colors to device colors
cannot in general be known at the time the application is running. To support requests
for arbitrary colors, a default lookup table that spans color space evenly (see (4)
below) will be loaded whenever possible. Colors specified explicitly in calls to
newplt(), **newpltw()**, **pencol()**, **penset()**, **color()**, and **regcol()** are stored literally in
the metafile and converted to color indexes only at rendering time, thus providing the
best available approximation to the requested colors. Some loss of color cannot be
avoided if the number of colors actually available is less than the number represented
in the metafile.

     (4) [This paragraph provides a method for dealing with obsolete displays that have
only a limited number of colors, and may not be implemented on newer systems.] The
default color lookup tables are constructed, when needed, with a sequence of bits for
each primary color in the order blue (high order), green, red (low order). Let n and r
be the quotient and remainder when the number of bits allocated for colors in the color
table is divided by three. If $r = 0$, allocate n bits for each color. If $r = 1$,
allocate n bits for blue and green, and $n + 1$ bits for red. If $r = 2$, allocate n bits
for blue, and $n + 1$ bits for green and red. The available range of values for each
color is then represented by these bits. This document is deliberately ambiguous as to
exactly what colors are provided for each value of each color bit string, as some expe-
rimentation is contemplated. In particular, it is desirable that equal blue, green, and
red values in the color index should map to gray levels on the device, that color index
0 should map to black and the maximum color index should map to white, and also that the
pure color specified by the maximum value for blue, green, or red alone should be as
bright as possible. However, all these conditions cannot be met simultaneously if r is
not zero.

     (5) The background of a plot is assumed to be black on visual display devices and
white on hard copy devices unless explicitly changed by the user (by drawing a filled
rectangle of the desired size before executing any other drawing commands). For conve-
nience, the color name "black" is translated to white when drawing on a display device,
but is not translated when drawing on a hard copy device. However, color index 0
(referenced in an **rcolor()** or **bitmap[s]()** call) is not translated, but draws black on
the display.

GRAPHICS METAFILE FORMAT

     The ROCKS plotting routines support generation of a graphics metafile, which may
contain data for one or more plots. Graphics metafiles may be viewed with the program
mfdraw, either online, while they are being produced, or offline. mfdraw uses OpenGL
and X graphics to render plots generated by applications using the graphics library.
mfprint is an obsolete SunOS program that performed this function on SunOS workstations.

To permit compact representation of plot information, a special-purpose binary metafile
has been defined rather than using any of the published standard file formats. To make
the binary encoding portable across machines with different internal architectures, it
is defined to use only fixed-point, not floating point, representations of numbers and
big-endian bit order within numbers. When a binary metafile must be transmitted across
a 7-bit communications link, a standard encoding method such as BinHex should be used.

Commands consist of a single operation code (an ASCII character truncated to the range
0x01-0x3f) followed by appropriate numeric and string parameters. All coordinates are
in inches (which may be rescaled by relative-scale devices as described above). Angles
specified in degrees are converted to fractions of a cycle in the metafile. All char-
acter strings are encoded in ASCII, even on systems that have other native character
codes (e.g. EBCDIC). (Strings to be plotted literally are passed to the library in the

native character set of the executing host and converted to ASCII by the plot library if
necessary).

The metafile provides for variable-width representations of coordinates. Coordinate
range defaults to 6 integer bits and resolution to 10 fraction bits but the total of
integer plus fraction bits can be increased up to 27 bits by the initial call to
setmf(). (The default is sufficient to specify the location of any dot on up to 32"
wide paper with a 600 dpi laser printer or a 1/720" resolution typesetter. Seven
fraction bits can be used when typical hard-copy plotter resolution (0.01") is
sufficient.)

Numeric parameters vary in length and format according to their types and magnitudes as
described next. Commands are packed into a continuous bit stream. In parallel com-
puters, special consideration is necessary to assure that plotting commands that may
overflow the length of a single buffer (bitmaps, polylines, text strings) are not
interrupted by a buffer from a different node. The program that collects metafile
buffers in a parallel computer recognizes these commands and refuses to accept input
from a different node until the command is complete. Short buffers may be written in
this case to complete a command without causing other nodes to wait for a full buffer,
which might be delayed. When a group of commands needs to start on an I/O unit (byte)
boundary, for example, at the start of a buffer sent from a computational node to mfsr,
or at the start of a block of commands to be saved for reuse, an ALIGN command is
inserted at the end of the previous command to force the reader to skip over any unused
bits before processing the next block of commands. Character strings within a single
command are also aligned on a byte boundary, but here an ALIGN command is not needed
because the unused bits are known not to constitute the start of a new command.

All bit values are encoded in "big-endian" format, that is, when the metafile is viewed
as a string of bits, the high-order bit of each numerical variable comes first (closest
to the start of the file) and the low-order bit of each variable comes last (closest to
the end of the file). Negative fixed-width integers are encoded as two's complements.
Software which implements reading or writing of binary metafiles on machines which use
other bit or byte orders must translate the data as required on input and on output to
meet this standard.

The following table indicates the encoding used for each type of parameter which may
occur in the graphics calls. A type indicator is also defined for each. This type
indicator is used in the metafile record description to indicate the type of each para-
meter, which in turn implies its recording format. 'lci' and 'lcf' respectively indi-
cate the user-requested number of bits for coordinate integers and coordinate fractions.

Many of the parameters have alternative short and long encoding forms. Metafile readers
must be able to read all alternative forms for each type of parameter; metafile generat-
ing routines should use the shortest encoding compatible with the data in each instance.
Some of the encodings for coordinate variables make use of the previous value of the
same variable. For this purpose, metafile handling routines are required to maintain a
state consisting of the most recent values of the five variables x, y, radius, width,
and height (designated by type indicators x,y,r,w,h, respectively) for each drawing
frame. These are used to expand the short-form coordinate codes. Type codes x',y'
indicate values that can be encoded relative to the current values of x,y, respectively,
but which do not themselves update these stored values. The stored values are invalid-
ated at the start of a new plot or local drawing frame, and, in parallel libraries, the
current window and frame numbers are rewritten at the start of each new metafile buffer,
so the drawing program will use the correct current state.

```
             TYPE
PARAMETER    INDICATOR    ENCODING
---------    ---------    ---------------------------------------
Coordinate    x,y    00 + lcf fraction bits.  Indicates that the given fraction should
                          be added to the previous parameter value of the same type to
                          give the new value.  A zero fraction is interpreted as an
                          increment of +1.0".

                     01 + lcf fraction bits.  Indicates that the given fraction
                          should be subtracted from the previous parameter value of the
                          same type to give the new value.  A zero fraction is inter-
                          preted as an increment of -1.0".
```

                        10. Indicates that the previous parameter value of the same
                        type is to be used.

                        11 + sign + lci integer bits + lcf fraction bits. This
                        encoding indicates a general coordinate value with negatives
                        expressed as the two's complement of the absolute value.

Size   r,w,h   Same as coordinates, except values are always positive and a
                        sign is not coded in the 11 form.

Angle   a   0 + 4 fraction bits. (Angle value is in cycles. This
                        notation can express angles that are commonly used for axis
                        directions in 1/16 cycle increments. Angles larger than one
                        cycle are truncated to the fractional value.)

                        1 + sign + 15 fraction bits. (Angle value in cycles. This
                        notation can express any angle with a precision of 2**(-15)
                        cycle, or less than one arc min.)

Character   c<n>   String of n 8-bit ASCII characters. 'n' may be a decimal
                        integer or the literal character 'n', which indicates that
                        the length of the string is given by a variable appearing
                        earlier in the same metafile record. Character strings are
                        always aligned on a byte boundary. An alignment record is
                        not needed when a string occurs within a command. (There is
                        currently no provision to encode wide Unicode characters.)

General   g   00. (Indicates parameter value is zero.)
                        010. (Indicates parameter value is +1.0.)
                        011. (Indicates parameter value is -1.0.)
                        10 + sign + 16 fraction bits.
                        110 + sign + 8 integer bits + 16 fraction bits.
                        111 + sign + 20 integer bits + 16 fraction bits.

                        This notation is used for general floating point values. It
                        permits a range of +/-1,000,000 and resolution of approx.
                        0.000015. If necessary, alternative routines will be pro-
                        vided to store single- and double-precision floating point
                        numbers as big-endian, 32- or 64-bit IEEE binary values,
                        respectively.

Bitmap value   j   1 bit for black-and-white bitmaps, lci bits for BM_COLOR
                        bitmaps, 8 bits for GM_GS grayscale or BM_C8 color bitmaps,
                        16 bits for BM_GS16 grayscale or BM_C16 color bitmaps, 24
                        bits for BM_C24 color bitmaps, or 48 bits for BM_C48 color
                        bitmaps.

Unsigned int   k   00 + 6 integer bits.
                        01 + 14 integer bits.
                        10 + 22 integer bits.
                        11 + 30 integer bits.
                        This notation is used for unsigned integer arguments such as
                        frame or object numbers.

Unsigned int   kn   Unsigned integer, exactly n bits.

Signed int   s   00 + sign + 5 integer bits.
                        01 + sign + 13 integer bits.
                        10 + sign + 21 integer bits.
                        11 + sign + 29 integer bits.
                        This notation is used for signed integer arguments.

A variable repetition count is indicated by placing the value in parentheses before the
format code, e.g. "(np)(x,y)" indicates "np" repetitions of x,y coordinate pairs.

<u>Metafile Data Records:</u>

     Each metafile begins with a header containing three ASCII-coded records. Formats
are indicated in FORTRAN style. Each header record is terminated with a newline char-
acter. The encoding of these records is compatible with old V1 ASCII-coded metafiles.

<u>Record 1: File identification and version number</u>

Format: 13HPLOTDATA V2A ,A1,2I3
Data: enc,lci,lcf
     enc = File encoding: A = ASCII (obsolete), B = binary as defined in this docu-
          ment. Code 'B' will be taken to imply that the byte width on the machine
          writing the file is 8 bits. Additional values of this code may be defined in
          the future to indicate new metafile codings.
     lci = length of coordinate integer portion in bits.
     lcf = length of coordinate fractions in bits.
Generated by: **newplt()** on first call only, but the 'lci' and 'lcf' parameters are set
     by an earlier call to **setmf()** or defaulted.
Note: V2A gives the version number of the metafile described in this document and will
     be updated when the specification changes. Versions prior to V1C do not have the
     enc, lci, or lcf fields. These files may be read by V1C-to-V2A translators if the
     values are taken to be enc = A, lcf = 10, lci = 0.

<u>Record 2: Application title information</u>

Format: A60
Data: title
     title = user-provided title information for current run.
Generated by: **newplt()** on first call only.
Note: The title field is obtained by calling **gettit()**.

<u>Record 3: Time stamp</u>

Format: 6A2
Data: year,month,day,hour,min,sec
Generated by: **newplt()** on first call only.
Note: The time stamp is obtained by calling **tstamp()**.

<u>Additional Header Information ('_') record</u>

The following record is placed between the traditional three header records and the
first "Start of Plot" record to pass additional one-time-only information to the plot
rendering program. A version number is provided to allow for future compatible expan-
sion of this record if needed. It is assumed that rendering programs will provide
suitable defaults if this record is missing.

Format: X'1F',k4,k16
Data: ahiv,nnodes
     ahiv = additional header information version. '0' refers to the version described
          here.
     nnodes = number of nodes used to create this file in a parallel computer. '0'
          indicates a serial computer. A rendering program can use this information to
          preallocate an array of structures to keep track of the graphics state in
          each node (x,y,r,h,w,color,line thickness,etc.).

Data for one or more plots follow the header. Each plot begins with a "Start of Plot"
record defined as follows:

<u>Start of Plot ('[') record:</u>

Format: X'1B',k,k,k,k,g,g,k,k,k,k6,cn
Data: window,frame,mfindex,xgindex,xsiz,ysiz,nexpose,movie_device,movie_mode,nc,chart
     window = window number where this plot should be placed. '0' refers to the
          initial, default window.

          frame = frame number assigned initially for use in this plot.
          mfindex = metafile plot index, counting from 1 in each file. (Note that prior to
               V1C, this was a 4 character field with a maximum value of 9999; in V2A it is
               a binary integer with max value 2**32-1, permitting over 4*10^9 plots in a
               file).
          xgindex = X graphics plot index. This may differ from 'mfindex' if different sets
               of plots are displayed than are written to the metafile.
          xsiz,ysiz = Plot size along x and y axes.
          nexpose = Number of exposures of this plot if making a (slow-motion) movie.
          movie_device = Code number for device used to make movie.
          movie_mode = Movie mode (BATCH, MOVIE, or STILL, see plotdefs.h)
          nc = number of characters in 'chart' string (max 32).
          chart = chart name. When plotting on a hard-copy device, this indicates the type
               of paper or other medium requested for this plot. On a graphics display
               device, a name other than 'DEFAULT' replaces the window title given in Record
               2. Implementations are free to ignore requests for chart types that they do
               not support. 'DEFAULT' is always supported and yields the default chart type
               (hard copy) or window title (graphics display device).
Generated by: **newplt()** and **newpltw()**
Notes: The xorg, yorg, and fmode arguments of **newplt()** or **newpltw()** are recorded separ-
     ately in frame definition records, and the pen type and pen color in Pentype and
     Pencolor records (see below). 'chart' is the corresponding argument to **newplt()** or
     newpltw(). Sufficient paper or other resources for a plot of the specified size
     is requested (absolute scale devices), or the overall scale is adjusted such that
     xsiz,ysiz in user coordinates will fit in the available plotting space with
     preservation of the original aspect ratio (relative scale devices). The current
     plotting frame is set to frame 0 (default window) or the frame established by
     newwin(). This frame always refers to the entire plot. The current plotting
     position is set to xorg,yorg and subsequent plotting coordinates are interpreted
     relative to this origin. The metafile and X graphics plot indexes are both
     written in order to simplify the buffer writing code, even though only one is
     relevant to each output medium.

Each plot may contain any desired plotting records constructed according to the follow-
ing definitions. Record formats are given in alphabetic order according to the initial
command letter before truncation. No special record formats are defined for number,
axis, and arrow-drawing commands--these are expanded in the metafile to plotting primi-
tives. Brackets enclose items that may be present or absent or that may have different
formats in different variants of the record.

New window record ('{'):

Format: X'3B,k
Data: iwin
     iwin = number of new window to be created.
Generated by: newwin() call, which assigns the new window number.
Note: In a parallel computer, this record is issued to the metafile only by Node 0.
     In- and out-bound synchronization is performed, which allows all nodes to be
     informed of the new assigned window number. This will generally just be the next
     integer higher than the previous window, but code should not assume this to be
     always true.

Reserved data record ('@'):

Format: X'00'
Data: This code is reserved for internal use by the plotting package and should never
     be assigned as a metafile data record type.

Arc record ('A'):

Format: X'01',k2,x,y,x',y',a
Data: type,xc,yc,xs,ys,angle

        type = the literal value '00' indicating a circular arc, '10' indicating a circu-
               lar arc from an **arc2()** call. The values '01' and '11' are reserved for
               future use to define elliptical arcs.
        xc,yc = x,y coordinates of center of circular arc.
        xs,ys = x,y coordinates of start of arc.
        angle = angular extent of arc (drawn counterclockwise from xs,ys if angle > 0,
               clockwise if angle < 0).
Generated by: **arc()** or **arc2()** call

Bitmap record ('B', unscaled):

Format: X'02',2k4,x,y,6k,[skip],(wd*ht)j
Data: type,mode,x1,y1,rowlen,colht,xoff,yoff,wd,ht,[skip],p11,p12,...,
        p21,p22,...,p(wd,ht)
        type = a hexadecimal digit encoding the type of bitmap. Its value is BM_BW (0) if
               the bitmap contains black-and-white data, BM_GS (1) if the bitmap contains
               8-bit grayscale data, BM_GS16 (2) if the bitmap contains 16-bit grayscale
               data, BM_C48 (3) if the bitmap contains 48-bit (16 bits per color) color
               data, BM_COLOR (4) if the bitmap contains indexed color data, BM_C8 (5) if
               the bitmap contains 8-bit (2-3-3 BGR) color data, BM_C16 (6) if the bitmap
               contains 16-bit (1-5-5-5 RGB) color data, or BM_C24 (7) if the bitmap
               contains 24 bit (8 bits per color) color data. Other values may be defined
               in future versions to indicate other color modes or bitmap compression. In a
               black-and-white bitmap, each pixel value is 0 to indicate background color or
               1 to indicate foreground color. In a grayscale bitmap, each pixel value is a
               number from 0 to 255 (65535), indicating a color from black (0) to white.
        mode = a hexadecimal digit indicating the drawing mode of the bitmap (see descrip-
               tion of **bitmap()** function). Values are GM_SET (0) to replace any existing
               color at each position of the bitmap with the bitmap color, GM_XOR (1) to
               "xor" the bitmap color with any existing color at each position, GM_AND (2)
               to "and" the bitmap color with any existing color at each position, and
               GM_CLR (3) to clear (set to background) each bitmap position at which the
               bitmap color index is not zero. The high order bits are reserved for future
               use and must be 0.
        x1,y1 = x,y display coordinates (in inches) of lower left-hand corner of full
               bitmap (of which this metafile record may correspond to only a portion).
               These values may be modified by pixel offsets xoff,yoff, respectively.
        rowlen,colht = width and height of the complete bitmap (not just the portion being
               plotted by this call) in pixels.
        xoff,yoff = positive x,y offsets (in pixels) of lower left-hand corner of this
               bitmap or bitmap fragment from the origin (x1,y1). These parameters may be
               set by a **bitmap()** function in a parallel computer to subdivide bitmaps into
               rectangular tiles plotted from different processors. 'xoff' is the same as
               the 'xoffset' parameter of **bitmap()**, but 'yoff' may differ from 'yoffset'
               depending on the row scanning order and buffering.
        wd,ht = width and height of this bitmap segment, in pixels (these parameters do
               not reset the w,h metafile variables). Alternatively, 'ht' maybe coded as 0,
               in which case 'wd' encodes the number of consecutive pixels in scan order
               contained in this record. [This is intended for use on parallel computers
               where the pixels in a bitmap are simply divided linearly among compute nodes
               without forming subrectangles of the whole map.]
        skip. In a black-and-white bitmap, a four-bit bit count field followed by enough
               zero bits to align the buffer on a byte boundary are inserted here. Use of
               the count field allows for the theoretical possibility that the reading
               program may have a different byte size than the writing program.
        p11,p12, etc. Pixel values defining a bit map. If the 'type' value is BM_BW,
               there is one bit per pixel. If the 'type' value is BM_GS or BM_C8, there is
               one byte per pixel. If the 'type' value is BM_GS16 or BM_C16, there are two
               bytes per pixel, if the 'type' value is BM_C24, there are three bytes per
               pixel, and if the 'type' value is BM_C48, there are six bytes per pixel.
               Pixels are arranged in "television" scanning order, horizontally from left to
               right and then vertically from top to bottom. In all encodings, each succes-
               sive row in memory is aligned on a byte boundary. The data transmitted for
               each row begin with the byte containing the 'xoff' pixel and end with the
               byte containing the 'xoff' + 'wd' - 1 pixel. Any unused bits in the first
               and last bytes must be discarded by the program reading the metafile. The

           bitmap() function may accept alternative orderings of bit maps, but these
must be rearranged to the standard order given here for recording in
metafiles.
Notes: (1) An unscaled bitmap is plotted with one pixel in the bitmap data mapped to one
pixel on the graphics display. This may be most useful for large bitmaps
where best performance is the issue. A scaled bitmap (see below) is scaled
to fit a specified width and height. These dimensions are given in "inches"
so vector objects can be accurately superimposed on the bitmap.
    (2) Note: A bitmap record may contain data for a rectangular fragment of a full
bitmap as a result of parallel computation or buffering. The fragments are
recombined during plotting. When a bitmap is divided among two or more
physical records in a parallel computer, the bitmap() routine is required to
provide a new bitmap record header at the start of each physical record (node
to host message) to avoid disruption of the bitmap by intervening data from
other nodes.
Generated by: **bitmap()** call
Note: Bitmaps may be ignored if the output device does not support them.

Bitmap record ('b', scaled):

Format: X'22',2k4,x,y,w,h,6k,[skip],(wd*ht)j
Data: type,mode,x1,y1,bwd,bht,rowlen,colht,xoff,yoff,wd,ht,[skip],p11,p12,...,
    p21,p22,...,p(wd,ht)
Notes: All data items same as for unscaled bitmap except 'bwd', 'bht' are added.
    bwd,bht = width and height of full bitmap (of rowlen columns and colht rows) in
        inches. The data are scaled to fit in the specified rectangular area. (The
        scales in the x and y directions are not necessarily equal.)
Generated by: **bitmaps()** call
Note: Bitmaps may be ignored if the output device does not support them.

Circle record ('C'):

Format: X'03',k2,x,y,r
Data: fill,xc,yc,radius
    fill = the literal value '00' to plot an outline circle or '01' to plot a filled
        circle. The value '10' is reserved for future use to define a pattern fill.
    xc,yc = x,y coordinates of center of circle.
    radius = radius of circle.
Generated by: **circle()** call
Note: Filled circles may be plotted as open circles if the output device does not
    support fills.

Draw progressive line record ('D'):

Format: X'04',x,y
Data: x,y
    x = x coordinate of point to be plotted.
    y = y coordinate of point to be plotted.
Generated by: **plot()** call with 'ipen' = PENDOWN
Notes: This record signifies that a line in the current color should be drawn from the
    current plotting position to the point x,y and then the current plotting position
    should be set to x,y. The line is drawn only once, regardless of the current
    retrace setting.

Ellipse record ('E'):

Format: X'05',k2,x,y,w,h,a
Data: fill,xc,yc,wd,ht,angle
    fill = the literal value '00' to plot an outline ellipse or '01' to plot a filled
        ellipse. Other values are reserved for future use to define pattern-filled
        ellipses.
    xc,yc = x,y coordinates of center of ellipse.
    wd = width of ellipse (half length of first axis).

```
        ht = height of ellipse (half length of second axis).
        angle = rotation of wd axis from positive x axis.
Generated by:  ellips() call
Note: Tilted ellipses may also be generated by using a tilted frame.  Filled ellipses
      may be plotted as open ellipses if the output device does not support fills.
```

Font and register font record ('F'):

```
Format:  X'06',k,k6,cn
Data: jf,n,font
        jf = 0 if this record was generated by a font() call or a positive integer if this
              record was generated by a regfont() call, in which case it is the font number
              assigned to this font.
        n = number of characters in font name (max 40).  If jf = 0 and n = 0, the 'font'
              parameter is omitted and subsequent text plotting is done using the default
              stroke font.  If jf != 0, n = 0 is an error.
        font = name of font to be used for subsequent text plotting.
Generated by:  font() or regfont() call
Note:  Font support is device dependent.  The font name 'DEFAULT' is always available
       and causes the program to plot stroke text using an internal font which produces
       lettering with the same predictable size and aspect ratio on all devices (char-
       acter width = (4/7) * height, spacing = (6/7) * height).  The default font is
       initially active.  Furthermore, any font request which is not recognized by the
       implementation is mapped to the default font.
```

Registered font record ('f'):

```
Format:  X'26',k
Data: rfont
        rfont = font number returned by a previous call to regfont().  This font will be
              used for subsequent text plotting in the current window.  If rfont = 0,
              subsequent text plotting is done using the default stroke font.
Generated by:  rfont() call
Note: Same as for Font record.
```

Polygon/star record ('G'):

```
Format:  X'07',k3,x,y,r,g,g,k,a
Data:  fill,xc,yc,rv,indent,spike,nv,angle
      fill = the literal value '000' to plot a disconnected outline polygon, '001' to
            plot a disconnected, filled polygon, '010' to plot an outline polygon with a
            connecting line from the old current drawing position to the center coordi-
            nates (xc,yc), or '011' to plot a connected, filled polygon.  (The first bit
            of the binary codes is reserved for future use and should always be 0.)
      xc,yc = x,y coordinates of center of polygon.
      rv = radius of polygon as measured at any vertex.
      indent = 0 if the sides of the polygon are to be straight lines, otherwise, a
            factor to be multiplied by 'rv' to determine the radius at the center of each
            side of the polygon.
      spike = 0 if the vertices of the polygon are to be undecorated, otherwise, a
            factor to be multiplied by 'rv' to determine the radius at the end of a spike
            to be drawn at each vertex of the polygon.
      nv = number of vertices.
      angle = angle by which first vertex is rotated from horizontal.
Generated by:  polygn() call
```

Retrace/thickness record ('H'):

```
Format:  X'08',k4
Data:  krt
      krt = amount of thickening of lines in following objects (0 <= krt <= 15).
Generated by:  retrace()
```

Note: The default value of krt is 0, indicating that each open object is drawn at
       standard line thickness. Larger values indicate that the desired total line width
       is approximately 0.01*krt inches. On pen plotters, each line is redrawn 'krt'
       times to produce a thicker line. Command code "H" may be thought of as standing
       for "heaviness".

Explicit numeric color record ('I'):

Format: X'09',3k8
Data: blue, green, red
      blue, green, red = color values to be used for following plot commands, each
            encoded as an integer value from 0 (dimmest) to 255 (brightest).
Generated by: **color()** call

Registered color record ('i'):

Format: X'29',k
Data: jc
      jc = index used to set plotting color to value registered with a previous **regcol()**
            call
Generated by: **rcolor()** call

Pen color record ('K'):

Format: X'0B',k4,cn
Data: nc,cname
      nc = number of characters in color name (max 15). If nc is 0, there is no 'cname'
            and the color is set to "BLACK"
      cname = name of color to be used next.
Generated by: **newplt()**, **newpltw()**, **pencol()**, or **penset()** call
Note: cname may be the English name of a color, e.g. "RED", or it may be a hexadecimal
       expression defining the desired color as follows: Colors may be given as twelve-
       bit values (4 bits each for blue, green, red) or as twenty-four bit values (8 bits
       for each primary color). Twelve-bit colors are expressed as "Xbgr" where b, g,
       and r are single hexadecimal digits expressing the blue, green, and red values
       respectively on a scale where 'F' (15) is the maximum value for each color.
       Twenty-four bit colors are expressed as "Zbbggrr" where the two hexadecimal digits
       give an eight-bit value for each color on a scale where 'FF' (255) is the maximum
       value for each color. Examples are "X5FF" or "Z86A43E". The routine that
       interprets the metafile is supposed to select the nearest available color that
       matches the requested color.

Register color record ('k'):

Format: X'2B',k,k,cn
Data: jc,nc,cname
      jc = synonym index to be assigned to this color.
      nc = number of characters in color name.
      cname = name of color--see notes above for pen color record.
Generated by: **regcol()** call

Line record ('L'):

Format: X'0C',x,y,x,y
Data: x1,y1,x2,y2
      x1,y1 = x,y coordinates of beginning of line.
      x2,y2 = x,y coordinates of end of line. The line is drawn using the current
            retrace-thickness setting. The current position is set to (x2,y2) after the
            line is drawn.
Generated by: **line()** call

                                 PLOTTING ROUTINES

<u>Thin line record ('l'):</u>

Format: X'2C',x,y,x,y
Data: x1,y1,x2,y2
     x1,y1 = x,y coordinates of beginning of line.
     x2,y2 = x,y coordinates of end of line. A hair-thin line is drawn, ignoring the
          current retrace-thickness setting. The current position is set to (x2,y2).
Generated by: **plot()** call with ipen = PENDOWN

<u>Move record ('M'):</u>

Format: X'0D',x,y
Data: x,y
     x = x coordinate of point to be set as current plot location.
     y = y coordinate of point to be set as current plot location.
Generated by: **plot()** called with 'ipen' = PENUP
Note: This record signifies that the current plotting position should be moved to point
     x,y without drawing anything.

<u>Object identifier record ('O'):</u>

Format: X'0F',k
Data: id
     id = identification number of object drawn by following plot commands until a new
          id number is encountered.
Generated by: **gobjid()** call

<u>Polyline record ('P'):</u>

Format: X'10',k3,k,(np)(x,y)
Data: fill,np,x1,y1,x2,y2,...,x(np),y(np)
     fill = the literal value '000' to plot an open, hairline thin polyline, '001' to
          plot an open (linear), thick polyline, '010' to plot a closed, thin polyline,
          '011' to plot a closed, thick polyline, or '111' to plot a closed, filled
          polyline. The value '100' is reserved for future use to define a pattern
          fill. With codes 010, 011, and 111, but not with codes 000 and 001, the last
          point in the polyline is joined to the first point to form a closed figure.
     np = number of points in the polyline.
     x1,y1, etc. x,y coordinates of np points used to form an open polyline or closed
          polygonal figure.
Generated by: **polyln()** call
Note: Filled closed polylines may be drawn as outlines if filling is not supported by
     the hardware or underlying rendering library.

<u>Square record ('Q'):</u>

Format: X'11',k2,x,y,r
Data: fill,x1,y1,size
     fill = the literal value '00' to plot an outline square or '01' to plot a filled
          square. The value '10' is reserved for future use to define a pattern fill.
     x1,y1 = x,y coordinates of lower left-hand corner of square.
     size = edge of square.
Generated by: **square()** call. (This record is also generated by **rect()** when the two
     edges of the rectangle are equal).

<u>Rectangle record ('R'):</u>

Format: X'12',k2,x,y,w,h
Data: fill,x1,y1,wd,ht
     fill = the literal value '00' to plot an outline rectangle or '01' to plot a
          filled rectangle. Other values are reserved for future use to define
          pattern-filled rectangles.

        x1,y1 = x,y coordinates of lower left-hand corner of rectangle.
        wd = width of rectangle.
        ht = height of rectangle.
Generated by: **rect()** call

Symbol record ('S'):

Format: X'13',x,y,h,a,s,cn
Data: x,y,ht,angle,n,text
        x,y = coordinates of lower left corner of plotted text (n > 0) or center of
            plotted symbol (n <= 0).
        ht = height of plotted text.
        angle = rotation angle of text baseline from horizontal.
        n = length of text in characters. Negative and zero values are permitted and are
            interpreted as defined in the description of **symbol()**.
        text = symbol or text string to be plotted.
Generated by: **mfprtf()**, **symbol()**, or **number()** call.
Note: When used with the default font, the current x,y coordinates are updated to the
      location at the end of the string where a continuation string would begin, allow-
      ing use of **symbc()** or **numbc()** for the continuation string. With other fonts, the
      string width cannot be calculated; **mfprtf()** should be used to combine all string
      components into a single Symbol record.

Continued symbol record ('s'):

Format: X'33',h,a,s,cn
Data: ht,angle,n,text
      ht = height of plotted text.
      angle = rotation angle of text baseline from horizontal.
      n = length of text in characters. Negative and zero values are permitted and are
          interpreted as defined in the description of **symbol()**.
      text = text string to be plotted.
Generated by: **symbc()** or **numbc()**.

Pen type and register pen type record ('T'):

Format: X'14',k,k4,cn
Data: jt,n,ptype
      jt = 0 if this record was generated by a **newplt()**, **newpltw()**, **penset()**, or
           pentyp() call or a positive integer if this record was generated by a
           regpen() call, in which case it is the number assigned to this pen type.
      n = number of characters in pen type name (max 15). If jt = 0 and n = 0, the
          'ptype' parameter is omitted and subsequent plotting is done using the
          default full line pen type. If jt != 0, n = 0 is an error.
      ptype = string describing a particular "pen type".
Generated by: **newplt()**, **newpltw()**, **penset()**, **pentyp()** or **regpen()**.
Notes: The allowed values of 'ptype' depend on the eventual output device, for example
      "FELTTIP" for a pen plotter. The special names "DOT", "DASH", "DOT-DASH", and
      "DASH-DOT" will be implemented where possible on visual displays to specify
      dotted, dashed, etc. lines. An implementation is free to ignore pen types it does
      not support.

Registered pen type record ('t'):

Format: X'34',k
Data: rpen
      rpen = pen type number returned by a previous call to **regpen**(). This pen type
             will be used for subsequent plotting in the current window. If rpen = 0,
             subsequent text plotting is done using the default full line pen type.
Generated by: **rpen()** call
Note: Same as for Pen type record.

Viewpoint (local frame) definition record ('V'):

Format: X'16',k10,k,w,h,[6g]
Data: modes,vn,width,height,[vxx,vxy,vxc,vyx,vyy,vyc]
      modes = Sum of any of the following defined bits used to control modes of
            operation of this drawing frame: FRM_BORD (0x01), has border; FRM_CLIP
            (0x02), clip at edges; FRM_DRAG (0x04), frame is draggable by user; FRM_ROTN
            (0x08), rotation/translation matrix is entered; FRM_STOR (0x10), commands are
            stored for later use; FRM_BMDR (0x20), frame is bookmarked each time drawn;
            FRM_SYNC (0x40), **frmdef()** requires outbound sync; FRM_BDIS (0x80), **frmdbm()**
            requires inbound sync. Bits 0x0100 and 0x0200 are reserved for future use.
      vn = Viewpoint (local frame) number. Used with all action codes. (If transform
            'vn' has not been defined, drawing is referred to the default axes,
            corresponding to vn = 0. The active frame cannot be deleted.)
      width,height = width and height of outline box in coordinate system frame that is
            current after the new definition takes effect.
      vxx,vxy,vxc,vyx,vyy,vyc = coefficients used to determine x(plotted) and y(plotted)
            from x and y application coordinates according to:
                  x(plot) = x*vxx + y*vxy + vxc.
                  y(plot) = y*vyx + y*vyy + vyc.
            Included only with action code FRM_ROTN.
Generated by: **frmdef()** call.
Notes: Coordinate transformations for each defined local frame are saved in the
      rendering program and their lifetime extends across plots until the frame is
      deleted or the program terminates. In some implementations, the user may be able
      to modify these transformations by dragging the frame with a pointing device.
      When a frame is deleted, it can no longer be made active nor can it be resized or
      repositioned. Its outline box and any objects drawn in the deleted frame remain
      present until the start of the next plot. Origin shifts generated by calls to
      plot() apply only to the current drawing frame and only on the current processor
      node.

Viewport (local frame) change record ('v'):

Format: X'36',k
Data: ifrm
      ifrm = number of frame into which following drawing commands are to be entered.
Generated by: **frame()** call.

Viewport (local frame) update record ('u'):

Format: X'35',k3,k
Data: fac,ifrm
      fac = action to be performed on frame 'ifrm': the literal value '001' to copy the
            stored plot commands to the current window; '010' to update the display with
            all plot commands since the last bookmark, then make a new bookmark; '011' to
            delete the stored plot commands; '100' to copy the frame to the push-down
            stack; '101' to pop the contents of the pushd-down stack into the current
            stack.
      ifrm = number of previously defined frame to be acted upon (ignored and coded as 0
            for the push and pop frame commands, but cannot be 0 for the other commands
            because frame 0 is the default frame and cannot be the subject of any of
            these actions).
Generated by: **frmplt()**, **frmdbm()**, **frmdel()**, **frmpush()**, or **frmpop()**.

Window control record ('W'):

Format: X'17',k2,k
Data: action,iwin
      action = the literal string '01' to draw (or redraw) the selected window; and the
            literal string '11' to delete the selected window.
      iwin = number of the window to be acted upon.
Generated by: **updwin()** or **clswin()**.

Window selection record ('w'):

Format: X'37',k
Data: iwin
     iwin = window number. Following drawing commands are drawn in the numbered window
            until another window selection record is encountered. Window 0 is always the
            default window (the first window opened). The plot library will generate an
            error if an undefined window is selected.
Generated by: **chgwin()**.

Graphics mode record ('X'):

Format: X'18',k4
Data: gm
     gm = graphics drawing mode: sets binary function used to combine current drawing
            color with any existing color at each pixel affected by following graphics
            commands. 0 indicates GM_SET mode (draw in the current color, obliterating
            any previous color), 1 indicates GM_XOR mode (colors are combined with XOR
            operation), and 2 indicates GM_AND mode (colors are combined with AND opera-
            tion). Other values are reserved for future use. Default is GM_SET mode.
Generated by: **gmode()** call
Note: Metafiles containing this command cannot be rendered on pen plotter devices.

Note: Command codes Y and Z are reserved for future use for hither-yon clipping and
Bezier splines, respectively. Only J and N uppercase command letters are currently
unused.

Alignment (no operation) record ('\' or '|'):

Format: X'1C',k4 or X'3C',k5
Data: nskip
     nskip = number of bits to skip following the X'1C' command code and the 4-bit skip
            count. This code is used on a parallel computer to skip over unused bits at
            the end of a buffer when another full command will not fit. This ensures
            that the reading program remains synchronized even if a buffer with different
            alignment is inserted from another node. (Note: In the unlikely event that
            the system has a byte size larger than 8, command code '3C' can be used,
            indicating that the skip count is a 5-bit integer field.)

State restoration record ('^'):

Format: X'1E',k,k,k
Data: node,iwin,ifrm
     node = number of node sending this buffer.
     iwin = current drawing window.
     ifrm = current drawing frame.

End-of-metafile record (']'):

Format: X'1D'
Generated by: **endplt()** call.
Note: The metafile always ends with an End-of-metafile record.

PLOTTING ROUTINES

Header Files for plotting

     The plot routines are prototyped in plots.h. plots.h #includes a second file,
plotdefs.h, which #defines certain named constants used in plot library calls as well
as in **mfdraw.** These constants are:

```
#define DFLTFRAME    0     /* Default plotting frame        */
```

                                  PLOTTING ROUTINES


```
#define FRM_BORD  0x01    /* Frame has border                */
#define FRM_CLIP  0x02    /* Clip objects at border          */
#define FRM_DRAG  0x04    /* Frame is draggable by user      */
#define FRM_ROTN  0x08    /* Rotation matrix is entered      */
#define FRM_STOR  0x10    /* Store commands for later use    */
#define FRM_BMDR  0x20    /* Bookmark whenframe is drawn     */
#define FRM_SYNC  0x40    /* frmdef requires outbound sync   */
#define FRM_BDIS  0x80    /* frmdbm requires inbound sync    */


#define PENDOWN     2     /* Move with pen down              */
#define PENUP       3     /* Move with pen up                */
#define RETRACE     1     /* Draw with retracing             */
#define NORETRACE   0     /* Draw without retracing          */
#define FILLED     -1     /* Draw filled object              */
#define THIN        0     /* Draw thin lines                 */
#define THICK       1     /* Draw thick lines                */
#define CLOSED_THIN  2    /* Draw closed thin lines          */
#define CLOSED_THICK 3    /* Draw closed thick lines         */
#define CENTERED    0     /* Center text at current x,y      */
#define VXX         0     /* Index of vxx in frame matrix    */
#define VXY         1     /* Index of vxy in frame matrix    */
#define VXC         2     /* Index of vxc in frame matrix    */
#define VYX         3     /* Index of vyx in frame matrix    */
#define VYY         4     /* Index of vyy in frame matrix    */
#define VYC         5     /* Index of vyc in frame matrix    */
#define AX_TKCCW    1     /* Plot ticks counterclockwise     */
#define AX_TKEFMT   2     /* E format tick values            */
#define AX_TKEXP    2     /* Log plot ticks as exponents     */
#define AX_LBLLS    8     /* Polar plot label long spokes    */
#define AX_LBLTOP  16     /* Polar plot label at top         */
#define BM_BW       0     /* Bitmap is black & white         */
#define BM_GS       1     /* Bitmap is 8-bit grayscale       */
#define BM_GS16     2     /* Bitmap is 16-bit grayscale      */
#define BM_C48      3     /* Bitmap is 16-16-16 RGB colored*/
#define BM_COLOR    4     /* Bitmap is index colored         */
#define BM_C8       5     /* Bitmap is 2-3-3 BGR colored     */
#define BM_C16      6     /* Bitmap is 5-5-5 RGB colored     */
#define BM_C24      7     /* Bitmap is 8-8-8 RGB colored     */
#define BM_NSME     8     /* Bitmap no submap extraction     */
#define GM_SET      0     /* Set pixels from current color   */
#define GM_XOR      1     /* XOR pixels with current color   */
#define GM_AND      2     /* AND pixels with current color   */
#define GM_CLR      3     /* Clear existing pixel data       */
#define MD_MATRIX   0     /* Movie device is MATRIX          */
#define MD_BEEPER   1     /* Movie device is beeper          */
#define MM_NOCHG    0     /* Movie mode no change            */
#define MM_STILL    1     /* Enter still mode                */
#define MM_MOVIE    2     /* Enter movie mode                */
#define MM_BATCH    3     /* Enter batch mode                */


#define BUT_INTX    1     /* Interrupt button was pressed    */
#define BUT_QUIT    2     /* Quit button was pressed         */
#define SKP_META    4     /* Skip metafile this frame        */
#define SKP_XG      8     /* Skip X graphics this frame      */
#define DO_MVEXP   16     /* Do movie exposure               */
```


A global data structure (currently called _RPG) contains state variables used by the
ROCKS plotting routines. This structure should be of no concern to the user.

DESCRIPTION OF THE PLOT CALLS

        Unless otherwise specified in the following descriptions, all angles are in deg-
rees. Angles are converted to cycle fractions for inclusion in the metafile. X extends
to the right and y extends upward from a specified origin (but successive rows in a bit
map are plotted from top to bottom). Coordinates are in arbitrary units, but any plot-
ting outside the limits (xsiz,ysiz) set by the call to **newplt()** or **newpltw()** may be
clipped. For output to devices that maintain an absolute plotting scale, coordinates
are in inches. With respect to retrace control, some implementations may draw
progressively thicker lines as the positive value of the 'krt' parameter increases (up
to a maximum value of 15), while others may simply provide thick and thin lines.

Whenever possible, invalid argument values produce some reasonable default action.
Cases that cause program termination are specifically noted in the individual function
writeups.

When programming on a parallel computer, note that **newplt(), newpltw(), newwin(),
updwin(), clswin(), finplt()**, and **endplt()** must be called in parallel from all nodes.
These routines result in node synchronization. New coordinate frames can be defined by
calling **frmdef()**, which has a mode bit to indicate whether the frame is local to that
node or should be called in parallel from all nodes with synchronization. Routines
setmf() and **setmovie()** should be called from node 0 only. The other routines, including
origin changes made with **plot()**, work independently on each node.

The plotting routines must maintain a state consisting of at least the current window
and frame numbers, coordinate transformation, current plotting position, height, width,
and radius, as these are used by the binary packing routines to select the most compact
numerical representations. Additionally, gmode, color, object identifier, line thick-
ness, character font, and pen type should be kept in order that routines that change
these parameters can detect that a call simply repeats the current value. All of these
variables are reset to their default values (or call argument values) by **newplt()** and
newpltw(). When each plotting function returns, the current plotting position is set to
the last point plotted (or some other point as specified in the description of the
individual routine given below). Plotting begins at the current plotting position when
plot() is called.

The plot library functions assume that the plot rendering routine (**mfdraw**) will also
maintain copies of these variables for each frame and window, so that when a new buffer
is initiated on a computational node, it is only necessary to write the node number,
window and frame number ("State restoration record") to the metafile in order for the
drawing routine to retrieve the appropriate state variables. Because colors set by
pencol() and **rcolor**() calls may be mapped differently to device colors when a metafile
is interpreted, parallel libraries must keep track on each node of which color-setting
routine was called most recently, and must use the same method to reset the current
color when a new buffer is initiated.

Subroutines **arc** and **arc2**

        These routines draw a circular arc in the current color and thickness. The only
difference is in the current plot position on return.

Usage:
        void **arc**(float xc, float yc, float xs, float ys, float angle)
        void **arc2**(float xc, float yc, float xs, float ys, float angle)

Arguments: (xc,yc) give the x,y coordinates of the center of the circular arc.

        (xs,ys) give the starting point from which the arc is drawn. These
    parameters implicitly define the radius.

        'angle' gives the angular length of the arc. If 'angle' > 0, the arc is
    drawn counterclockwise from (xs,ys). If 'angle' < 0, the arc is drawn clockwise
    from (xs,ys).

Current plot position on return: (xc,yc) if **arc()** was called; position of the end of
    the arc if **arc2()** was called.

Subroutine **arrow**

     This routine draws an arrow from (x1,y1) to (x2,y2) in the current color and
thickness.

Usage: void **arrow**(float x1, float y1, float x2, float y2, float barb, float angle)

Arguments: (x1,y1) and (x2,y2) are the coordinates of the two points between which an
     arrow is to be drawn.

          'barb' is the length of the arrow barbs.

          'angle' is the angle between the central line and the barb lines in degrees.
     Angles greater than 90 degrees will produce an arrow tail rather than an arrow
     head.

Current plot position on return: (x2,y2)

Subroutine **axlin**

     This routine is used to plot a linear axis in the current color and thickness with
tick marks at specified intervals. Subroutine **axlin()** labels the tick marks and plots a
label centered along the axis as a whole.

Usage: void **axlin**(float x, float y, const char *label, float axlen, float angle,
     float firstt, float deltat, float firstv, float deltav, float height, ind ktk,
     int nd, int nmti)

Arguments: 'x', 'y' are the coordinates of the starting point of the axis. Allow at
     least half an inch from the edge of the graph for the divisions and label.

          'label' is the label to be plotted along the axis. The height of the
     lettering is 1.33 times the 'height' parameter. If 'label' is NULL or points to
     an empty string or 'height' is <= 0, the label is not plotted.

          'axlen' is the length of the axis. If 'axlen' <= 0, nothing is plotted.

          'angle' is the angle at which the axis is plotted, measured in degrees
     counterclockwise from the positive x axis.

          'firstt' is the offset along the axis of the first tick mark in inches
     (normally 0).

          'deltat' is the offset of each successive major tick mark along the axis in
     inches. If 0, no tick marks or tick value labels are drawn.

          'firstv' is the value of the label for the first major tick mark.

          'deltav' is the increment added to 'firstv' for each successive major tick
     mark. If 0, no tick values are plotted.

          'height' is the height of the tick mark lettering (the axis label is plotted
     with letter height 1.33*height). If <= 0.0, tick mark values and the axis label
     are not drawn.

          'ktk' is the sum of any desired codes from the following list:
     AX_TKCCW (=1) Plot tick marks, tick values, and axis label counterclockwise from
          the axis (appropriate for the 'y' axis of a typical plot); default: plot tick
          marks and labels clockwise from the axis (appropriate for the 'x' axis of a
          typical plot).
     AX_TKEFMT (=2) Plot tick values in exponential format; default: use decimal
          format unless numerical value would require more than 12 digits to express.

          'nd' is the number of decimals in the tick mark labels. Use -1 to indicate
     that values should be plotted as integers, with no decimal point.

                                    PLOTTING ROUTINES

         'nmti' is the number of minor tick intervals (one more than the number of
    tick marks) to be plotted between labelled tick marks. If 0, no minor tick marks
    are plotted.

Current plot position on return: Undefined.

Subroutine **axlog**

         This routine is used to plot a logarithmic axis in the current color and thickness
    with tick marks at specified intervals. Subroutine **axlog()** labels the tick marks and
    plots a label centered along the axis as a whole.

Usage: void **axlog**(float x, float y, const char *label, float axlen, float angle,
    float firstt, float deltat, float firstv, float vmult, float height, int ktk,
    int nd, int nmti)

Arguments: 'x', 'y' are the coordinates of the starting point of the axis. Allow at
    least half an inch from the edge of the graph for the divisions and label.

         'label' is the label to be plotted along the axis. The height of the
    lettering is 1.33 times the 'height' parameter. If 'label' is NULL or points to
    an empty string or 'height' is <= 0, the label is not plotted.

         'axlen' is the length of the axis. If 'axlen' <= 0, nothing is plotted.

         'angle' is the angle at which the axis is plotted, measured in degrees
    counterclockwise from the positive x axis.

         'firstt' is the offset along the axis of the first tick mark in inches
    (normally 0).

         'deltat' is the offset of each successive major tick mark along the axis in
    inches. If 0, no tick marks or tick value labels are drawn.

         'firstv' is the value of the label for the first major tick mark.

         'vmult' is the amount that multiplies the value of each successive major tick
    mark (base of the logarithms).

         'height' is the height of the tick mark lettering (the axis label is plotted
    with letter height 1.33*height). If <= 0.0, tick mark values and the axis label
    are not drawn.

         'ktk' is the sum of any desired codes from the following list:
    AX_TKCCW   (=1) Plot tick marks, tick values, and axis label counterclockwise from
         the axis (appropriate for the 'y' axis of a typical plot); default: plot tick
         marks and labels clockwise from the axis (appropriate for the 'x' axis of a
         typical plot).
    AX_TKEFMT (=2) Plot tick values in exponential format; default: use decimal
         format unless numerical value would require more than 12 digits to express.
    AX_TKEXP   (=4) Plot tick values as base value ('vmult') to some power. In this
         case, 'firstv' should be the power indicated on the first tick, and 1 will be
         added to this exponent for each successive tick. Code AX_TKEFMT is ignored.

         'nd' is the number of decimals in the tick mark labels (ignored if AX_TKEXP
    is selected). Use -1 to indicate that values should be plotted as integers, with
    no decimal point.

         'nmti' is the number of minor tick intervals (one more than the number of
    tick marks, typically one less than the base) to be plotted between labelled tick
    marks. If 0, no minor tick marks are plotted.

Current plot position on return: Undefined.

Subroutine **axpol**

     This routine is used to plot a set of polar coordinate axes in the current color
and thickness. Axes may consist of circles, optional circle labels, short spokes, long
spokes with optional tick marks and labels, and an overall label placed above or below
the outermost circle.

Usage: void **axpol**(float x, float y, const char *label, float radius, float firstc,
     float deltac, float firstv, float deltav, float angle, float cvoffx, float cvoffy,
     float height, int ns, int nl, int ktk, int nd, int nmti)

Arguments: 'x', 'y' are the coordinates of the center point of the polar axes.

          'label' is a label to be plotted above or below the axis set (see 'ktk').
The height of the lettering is 1.33 times the 'height' parameter. If 'label' is
NULL or points to an empty string or 'height' is <= 0, the label is not plotted.

          'radius' is the radius of the polar diagram (defined as the radius of the
short spokes). If 'radius' <= 0, nothing is plotted.

          'firstc' is the radius at which the first full circle is plotted. Circles
are plotted at radial intervals 'deltac' until the next circle would exceed the
given radius.

          'deltac' is the radial increment between full circles. If 0, no circles or
circle value labels are drawn.

          'firstv' is the value of the label for the first full circle.

          'deltav' is the increment added to 'firstv' for each successive full circle.
If 'deltav' is 0, no circle value labels are plotted.

          'angle' is the angle (measured in degrees counterclockwise from the positive
x axis) of an imaginary radial line relative to which the circle value labels are
plotted. This parameter can be used to place the labels along or between spokes,
or even to have circle labels and no spokes.

          'cvoffx' is the x offset of the center of each circle label from the point of
intersection of its circle with the radius line defined by 'angle'.

          'cvoffy' is the y offset of the center of each circle label from the point of
intersection of its circle with the radius line defined by 'angle'.

          'height' is the height of the circle value and spoke angle lettering (the
axis label is plotted with letter height 1.33*height). If 'height' is zero,
circle values, spoke angles, and the axis label are not drawn.

          'ns' is the number of short spokes to be drawn around the circle. Short
spokes extend to the specified radius, have no tick marks between circles, and are
unlabelled. Short spokes that fall on the same locations as long spokes are not
drawn.

          'nl' is the number of long spokes to be drawn around the circle. Long spokes
extend to radius + 0.1*deltac, may have minor tick marks, and may be labelled
optionally with angles in degrees at their ends.

          'ktk' is the sum of any desired codes from the following list:
AX_TKEFMT (=2) Plot circle values in exponential format; default: use decimal
     format unless numerical value would require more than 12 digits to express.
AX_LBLLS   (=8) Label long spokes with angles in degrees.
AX_LBLTOP (=16)   Center axis label at top of plot; default: center label at
     bottom of plot.

          'nd' is the number of decimals in the circle labels. Use -1 to indicate that
values should be plotted as integers, with no decimal point.

        'nmti' is the number of minor tick intervals (one more than the number of
    tick marks) to be plotted between full circles on long spokes. If there are no
    circles, plot ticks from 'firstc' to 'radius'. If 0, no minor tick marks are
    plotted.

Current plot position on return: Undefined.

Subroutines **bitmap** and **bitmaps**

        These subroutines plot all or parts of bitmaps or raster arrays. bitmap()
plots the raster data directly, one display pixel per array element. bitmaps() scales
the bitmap to a specified size, measured in the same "inches" coordinates as vector
objects, so that image and vector objects can be accurately superimposed.

Usage:
    void **bitmap**(const byte *array, int rowlen, int colht, float xc, float yc,
        int xoffset, int yoffset, int iwd, int iht, int type, int mode)
    void **bitmaps**(const byte *array, int rowlen, int colht, float xc, float yc,
        float bwd, float bht, int xoffset, int yoffset, int iwd, int iht, int type,
        int mode)

Arguments: 'array' is a pointer to a byte array containing the data to be plotted.
    Data for each horizontal row must be packed in left-to-right order, with one bit
    per pixel if the 'type' is BM_BW, one byte per pixel if the 'type' is BM_GS or
    BM_C8, two bytes per pixel if the 'type' is BM_GS16 or BM_C16, three bytes per
    pixel if the 'type' is BM_C24, 6 bytes per pixel if the 'type' is BM_C48, or 'lci'
    bits per pixel if the 'type' is BM_COLOR. Rows are arranged in memory from top to
    bottom if 'iht' > 0, and from bottom to top if 'iht' < 0.

        'rowlen' and 'colht' give the size of the complete bitmap (not just the
    portion being plotted by this call) in pixels. The length of each row in bytes
    implied by 'rowlen' may be greater than or equal to the row length implied by the
    'type', 'xoffset', and 'iwd' parameters.

        'xc' and 'yc' are the coordinates in inches of the lower left-hand corner of
    the complete bitmap relative to the current plot origin. These coordinates are
    adjusted during plotting according to the values of 'xoffset', 'yoffset' for this
    bitmap fragment.

        'bwd' and 'bht' are the width and height of the full bitmap (i.e. a map of
    'rowlen' pixels per row and 'colht' pixels per column) in inches (**bitmaps()** only).

        'xoffset' and 'yoffset' are positive x,y display offsets (in pixels) of this
    bitmap fragment relative to the full bitmap specified by 'rowlen' and 'colht'.
    These parameters allow bitmaps to be plotted in subtiles from different computa-
    tional nodes in a parallel computer. If the BM_NSME ("bitmap no submap extrac-
    tion") bit is specified in the 'type' parameter, then the data array ("array"
    argument) contains only the information to be plotted by this call. In the
    default case (BM_NSME not specified), the argument array contains the entire
    bitmap and the data to be plotted are extracted from this full bitmap according
    to the 'xoffset', 'yoffset', 'iwd', and 'iht' parameters. ('yoffset' counts from
    the top of the bitmap if iht > 0 and from the bottom if iht < 0.)

        'iwd' and 'iht' are the width and height in pixels of the bitmap or rect-
    angular portion of a bitmap to be plotted by this call. Alternatively, 'iht' may
    be encoded as 0, in which case 'iwd' is the number of pixels contained in the data
    'array' arranged linearly beginning at the point given by 'xoffset','yoffset'.
    [This is intended for use on parallel computers where the pixels in a bitmap are
    simply divided linearly among compute nodes without forming subrectangles of the
    whole map.] If 'iht' > 0, the data in 'array' are arranged in top-to-bottom
    order; if 'iht' < 0, the data are arranged bottom-to-top. (Data are always stored
    in the metafile in top-to-bottom order.)

        'type' - is BM_BW (0) if the bitmap contains black-and-white data, BM_GS (1)
    if the bitmap contains 8-bit grayscale data, BM_GS16 (2) if the bitmap contains
    16-bit grayscale data, BM_C48 (3) if the bitmap contains 48-bit (16 bits per

     color) color data, BM_COLOR (4) if the bitmap contains indexed color data, BM_C8
     (5) if the bitmap contains 8-bit (2-3-3 BGR) color data, BM_C16 (6) if the bitmap
     contains 48-bit (16 bits per color) color data, or BM_C24 (7) if the bitmap
     contains 16 bit (5 bits per color, leftmost bit ignored) color data. Add BM_NSME
     (8) to this value if submap extraction is not to be performed on the "array"
     argument. Other 'type' values may be defined in future versions of this library
     to specify other color modes or bitmap data compression.

          'mode' is GM_SET (0) to set the color index at each point of the bitmap
     unconditionally from the array data, GM_XOR (1) to "xor" the array data with any
     existing data at each point, GM_AND (2) to "and" the array data with any existing
     data at each point, and GM_CLR (3) to clear any existing data at points of the
     bitmap for which the data array is nonzero. Type BM_BW should generally be
     specified with mode GM_CLR, because all nonzero array values are equivalent in
     this case.

Current plot position on return: unchanged

Function **bmpxsz**

     This function returns the number of bytes needed to store one pixel in a color
mode described by the argument 'bmcm'. The current plotting position is unchanged and
nothing is written to the metafile output.

Usage: int bmpxsz(int bmcm)

Argument: 'bmcm' is one of the bitmap types defined in the calls to bitmap() and
           bitmaps() above and defined in plotdefs.h with names of the form 'BM_xxxx'.

Subroutine **chgwin**

     This subroutine changes the current drawing window. Subsequent plotting commands
on this node draw in this window until a **newplt()**, **newpltw()**, or another **chgwin()** call
is issued. The window must have been defined by a previous call to **newwin()** or a
terminal error is issued.

Usage: unsigned int **chgwin**(unsigned int iwin)

Arguments: 'iwin' is the number of the window where drawing is to occur.

Return value: chwin() returns the number of the window in which drawing was occurring
     before the call. This may be used to restore drawing to an earlier window.

Current plot position on return: Whatever it was when last drawing in that window.

Subroutine **circle**

     This subroutine plots a circle in the current drawing color.

Usage: void **circle**(float xc, float yc, float radius, int kf)

Arguments: (xc,yc) are the coordinates of the center of the circle.

           'radius' is the radius of the desired circle.

           'kf' is a fill control switch. A filled circle is drawn if kf = -1 (FILLED).
     An unfilled circle is drawn if kf >= 0. The circle is thin if kf == 0 (THIN or
     NORETRACE), otherwise kf is taken as a line thickness parameter.

Current plot position on return: (xc,yc)

Subroutine **clswin**

     This subroutine closes the drawing window whose number is given and switches
drawing to the default window. Any attempt to close a window that was not previously
defined by a call to **newwin(),** or any attempt to change to a window that has already
been closed, results in a terminal error (abend code 208).

Usage: void **clswin**(unsigned int iwin)

Arguments: 'iwin' is the number of the window to be closed. When a window has been
     closed, the same window number may or may not be returned by a future **newwin()**
     call.

Current plot position on return: Whatever it was when last drawing in window 0.

Parallel computation: This routine must be called in parallel from all nodes; it results
     in inbound synchronization.

Subroutine **color**

     Subroutine **color()** changes the drawing color to a particular blue-green-red value.
This will generally be more efficient than a call to **pencol()** or **penset().** Routines
pencol() or **penset()** or the **regcol()**-**rcolor()** combination should be used if it is
desired to specify a particular color explicitly by name, for example, from a user-
entered parameter value.

Usage: void **color**(COLOR blue, COLOR green, COLOR red)

Arguments: 'blue', 'green', 'red' are the desired color values encoded as integers on a
     scale from 0 to 255 (which may be thought of as fixed-point values with eight
     binary fraction bits). The type 'COLOR' is declared in plotdefs.h.

Current plot position on return: unchanged

Parallel computation: On a parallel computer, each node can have its own current
     drawing color. Color changes affect only the node where executed.

Function **col2gr**

     This function returns the color mode obtained when a color mode from the set
accepted by the bitmap() or bitmaps() 'type' argument is averaged to produce the
grayscale equivalent. Basically, this routine gets the pixel length (1 or 2 bytes),
throwing away the color information. The current plotting position is unchanged and
nothing is written to the metafile output.

Usage: byte col2gr(byte bmcol)

Argument: 'bmcol' is the color mode whose gray (or single color) pixel size is to be
           extracted. It is one of the bitmap types defined in the calls to bitmap()
           and bitmaps() above and defined in plotdefs.h with names of the form
           'BM_xxxx'.

Function **deccol**

     This function decodes an ASCII drawing color from the set accepted by the pencol()
routine and returns its red, green, blue components. The current plotting position is
unchanged and nothing is written to the metafile.

Usage: int deccol(bgrv *pbgr, const char *pcol)

Arguments:'pbgr' is a pointer to a bgrv structre where the result is to be returned.
     'bgrv' is a typedef structure type declared in plots.h and contains 16-bit integer
     values for blue, green, red, respectively.

          'pcol' is a pointer to the ASCII color name to be decoded.

Value returned: 0 if color decoding was successful, 1 if the pcol argument was not an
     acceptable color name.

Subroutine **ellips**

     Subroutine **ellips()** draws an ellipse using the current color.

Usage: void **ellips**(float xc, float yc, float hw, float hh, float angle, int kf)

Arguments: (xc,yc) are the center coordinates of the desired ellipse.

          (hw,hh) are the half-width and half-height of the ellipse.

          'angle' is the counterclockwise rotation angle between the positive x axis
     and the axis defined by the 'hw' parameter.

          'kf' is a fill control switch. If kf = -1 (FILLED), a filled ellipse is
     drawn. An unfilled ellipse is drawn if kf >= 0. The ellipse is thin if kf == 0
     (THIN or NORETRACE), otherwise kf is taken as a line thickness parameter.

Current plot position on return: (xc,yc)

Note: Earlier versions of this routine did not have the 'angle' parameter.

Function **endplt**

     Function **endplt()** must be called at the end of every run in which plots are drawn
to close the device interface and/or plot output metafile.

Usage: int **endplt**(void)

Value returned: Same as for **newplt()**.

Parallel computation: This routine must be called in parallel from all nodes; it results
     in inbound synchronization.

Subroutine **factor**

     This routine enables the user to enlarge or reduce the size of an entire plot or
parts of a plot. All pen motions in the default drawing frame (including those used to
set a new plot origin) after the call to **factor()** and until the next call to **factor()**,
newplt(), or **newpltw()** are multiplied by the given scale factor. If a new origin has
been set prior to the call to **factor()**, its location is not affected. If **factor()** is
called before the first **newplt()** or **newpltw()** call in a run, all plotting frames in all
plots in the run are scaled.

Usage: void **factor**(float fac)

Argument: 'fac' is the scale factor to be used.

Current plot position on return: unchanged

Parallel computation: In a parallel computer, plotting is affected only on the
     processor node where the call is made.

Function **finplt**

     This function may be called to indicate to the plot library that all drawing com-
mands relating to the current plot in the default window have been completed. This
allows the plot to be produced on an online display even if **newplt()**, **newpltw()**, or
endplt() is not called immediately. In addition, **finplt()** can be called multiple times
during a long calculation to determine whether user interface buttons (QUIT or
INTERRUPT) have been activated.

Usage: int **finplt**(void)

Value returned: Same as for **newplt**().

Parallel computation: Call from all nodes. This provides synchronization of plot
     output.

Subroutine **font**

     This function specifies a font to be used by subsequent calls to **mfprtf()**,
simbol(), **simbc()**, **symbol()**, **symbc()**, **nomber()**, **nombc()**, **number()**, or **numbc()**. Font
names are device-dependent and so should be provided from user input.

Usage: void **font**(const char *fname)

Argument: 'fname' is the name of the new font (max 40 char). The name 'DEFAULT' is
     always accepted and switches to the default stroke font. A NULL pointer argument
     has the same effect.

Current plot position on return: unchanged

Parallel computation: On a parallel computer, each node can have its own font. Font
     changes affect only the node where executed.

Note: The limitation to 40 characters does not apply to registered font names.

Subroutine **frame**

     This subroutine causes subsequent plotting commands to be executed relative to a
previously defined local plotting frame.

Usage: void **frame**(unsigned int frame)

Argument: 'frame' is the number of the frame to be used. Frame 0 always refers to the
     default frame consisting of the entire plot in the current window. If 'frame'
     refers to a frame that has not yet been defined, frame 0 is used.

Current plot position on return: (0,0) in the new frame.

Parallel computation: On a parallel computer, the frame definitions for identically
     numbered frames must be the same on all nodes, because only one copy is stored on
     the metafile server. Frame selection affects only the node where executed.

Subroutine **frmdbm**

     This subroutine draws accumulated plotting commands for the specified local frame
from the last bookmark, then establishes a new bookmark after the last command plotted.
This routine can be used to update dynamically changing plots, such as neural network
raster plots, without redrawing the entire plot on each update. If material is to be
changed by the update, plotting modes such as GM_XOR should be used to erase the old
material.

Usage: void **frmdbm**(unsigned int frame)

Argument: 'frame' is the number of the frame to be drawn.

Current plot position on return: Unchanged.

Parallel computation: If the mode bit FRM_BDIS was set when the frame was created, all
nodes must call **frmdbm()** and an inbound sync is performed before drawing occurs. Other-
wise, any node may call for a redraw and the plot reflects whatever commands may have
been issued from the other nodes. This is useful when one node alone is executing the
plot calls involved in the update.

Subroutine **frmdef**

     This routine creates a local drawing frame and specifies its properties or changes
the coordinate transform of an existing frame. If plot commands are saved and an
outline box is drawn, it is drawn in the current color and that color is saved and used
each time the saved frame commands are copied to a plot window.

Usage: void **frmdef**(unsigned int *pfrm, unsigned int mode, float width, float height,
     [float vv[6]])

Arguments: 'pfrm' is a pointer to a frame number. If *pfrm is 0, a new frame is
     created in the current window and its number is returned in *pfrm. If *pfrm is
     greater than 0, it must contain the number of an existing frame. The width,
     height, and transformation matrix of this frame are changed to the specified new
     values, but the existing mode is unchanged, i.e. the 'mode' argument is ignored.

          'mode' is the sum of any of the following defined bits used to control modes
     of operation of this drawing frame: FRM_BORD (0x01), frame has border; FRM_CLIP
     (0x02), clip at edges; FRM_DRAG (0x04), frame is draggable by user; FRM_ROTN
     (0x08), rotation/translation matrix is entered; FRM_STOR (0x10), commands are
     stored for later use; FRM_BMDR (0x20), frame is bookmarked each time drawn;
     FRM_SYNC (0x40), **frmdef()** requires outbound sync; FRM_BDIS (0x80), **frmdbm()**
     requires inbound sync. Bits 0x0100 and 0x0200 are reserved for future use. The
     following combinations are implicit and are enforced by the code: FRM_DRAG
     requires FRM_SYNC and if a rotation/translation matrix is not provided, one is
     created with the identity transformation. FRM_STOR requires FRM_SYNC-the
     coordinate transformation must be the same on all nodes in a parallel computer.

          'width', 'height' give the width and height of the outline box in the coordi-
     nate system frame that is current after the new definition takes effect.

          'vv' gives the elements of the transformation matrix defining the local
     drawing frame in the order vxx, vxy, vxc, vyx, vyy, vyc, where x(plot) = x*vxx +
     y*vxy + vxc and y(plot) = y*vyx + y*vyy + vyc. The transformation matrix is
     included only with action code FRM_ROTN.

Current plot position on return: (0,0) in the new frame

Parallel computation: On a parallel computer, frame definitions are made available on
     all nodes if the FRM_DRAG, FRM_STOR, or FRM_SYNC bit is set. In this case, the
     same **frmdef()** call should be made on all nodes and an outbound sync occurs.

Restriction: At most 65535 frames can be created in any one run.

Subroutine **frmdel**

     This routine deletes the frame definition and stored commands for a local drawing
frame.

Usage: void **frmdel**(unsigned int frame)

Argument: 'frame' is the number of the frame to be deleted. If the FRM_STOR bit was
     not set for this frame, or the frame was already deleted, the **frmdel()** call is
     ignored-commands already transferred to a plot window cannot be deleted. When a
     frame has been deleted, the same frame number may or may not be returned by a
     future **frmdef()** call.

Current plot position on return: unchanged

Parallel computation: Frame deletions affect definitions and drawing on all nodes but
     only need to be executed on one node. If synchronization is required, call
     isynch() before **frmdel()** to assure that the deletion does not occur until all
     nodes have finished writing to the frame.

Subroutine **frminq**

        This routine obtains a copy of the coordinate transformation for a frame, which
may have been modified by user dragging.

Usage: int **frminq**(unsigned int frame, float vv[6])

Arguments: 'frame' is the number of the frame to be queried.

        'vv' is the coordinate transformation matrix, with the components in the same
        order as in the **frmdef()** call.

Value returned: The return integer is 1 if the user dragged the frame since the last
        frminq() call, otherwise 0. In either case, the current coordinate transformation
        matrix is returned in the area pointed to by the argument 'vv'.

Parallel computation: This inquiry may be executed from any node at any time. Note that
        even if a user drags a draggable frame, further plotting commands should still be
        entered in the defined original coordinate frame-the code on each node need not be
        aware of the new coordinate frame. The purpose of **frminq()** is to determine the
        new location of a frame so it can be saved in a suitable startup file so the frame
        will appear in the desired location the next time the application is run.

Subroutine **frmplt**

        This routine causes plot commands saved in a FRM_STOR frame to be copied to the
current plot window. If a new coordinate transformation matrix was entered via
frmdef(), it will apply to the saved plot commands retroactively.

Usage: void **frmplt**(unsigned int frame)

Argument: 'frame' is the number of the saved plot frame to be drawn.

Parallel computation: This command is executed on the plot server. It draws plot
        commands saved from all nodes, but only needs to be executed on one node. If
        synchronization is required, call **isynch()** before **frmplt()** to assure that drawing
        does not occur until all nodes have finished writing to the frame.

Subroutine **frmpop**

        This routine pops a saved plotting frame from a push-down stack. This allows a
coordinate system to be temporarily changed, then restored. An error occurs if the
push-down stack is emtpy, that is, there was no matching previous call to **frmpush()**.

Usage: void **frmpop**(void)

Subroutine **frmpush**

        This routine copies the current drawing frame onto a push-down stack, from which
it can be restored by a call to **frmpop()**. Typically, this will be followed by a call to
frmdef() to modify the frame's coordinate transformation.

Usage: void **frmpush**(void)

Subroutine **gmode**

        This function is used to change the drawing mode for all following graphics com-
mands except bitmaps, which have their own drawing mode parameter. XOR mode permits
limited animation through nondestructive erasing of objects already drawn. Use of a
drawing mode other than the default limits the portability of the application and should
be avoided.

Usage: void **gmode**(int mode)

Argument: 'mode' is GM_SET (0) to set the color index at each point drawn
     unconditionally to the current drawing color, GM_XOR (1) to XOR the current
     drawing color with any existing color index at each point, GM_AND (2) to AND the
     current drawing color with any existing data at each point, and GM_CLR (3) to
     clear any existing color at points where drawing occurs, regardless of the current
     drawing color. Additional modes may be defined in later versions of this library.

Default: GM_SET mode

Current plot position on return: unchanged

Parallel computation: On a parallel computer, each node can have its own current
     drawing mode. Mode changes affect only the node where executed.

Subroutine **gobjid**

     This routine is used to assign an arbitrary number to objects plotted by
subsequent plotting commands. In a future version of this library, the object number
may be returned when the user selects a particular object using a pointing device.

Usage: void **gobjid**(unsigned long id)

Argument: 'id' is an arbitrary positive integer, id < 2**30. This number is assigned
     to all objects drawn with subsequent drawing commands until a new object number is
     provided. Object id 0 may be used to stop assignment of the current id number to
     subsequent plotting commands.

Current plot position on return: unchanged

Parallel computation: On a parallel computer, each node can have its own current object
     id or multiple nodes can draw different parts of the same object. Object id
     changes affect only the node where executed.

Subroutine **gscale**

     This routine returns the absolute scale factor used to convert coordinates
provided to the plotting routines ("inches") into whatever internal units are used by
the plotting interface (e.g. pixels on a visual display). This value is not affected by
any change the user may make by dragging or resizing a local drawing frame.

Usage: float **gscale**(void)

Arguments: None

Return value: Scale factor as described above.

Current plotting position on return: unchanged

Parallel computation: The scale factor on the node executing the call is returned.

Subroutine **line**

     This routine draws a line from (x1,y1) to (x2,y2) in the current color and
thickness.

Usage: void **line**(float x1, float y1, float x2, float y2)

Arguments: (x1,y1) gives the x,y coordinates of the beginning of the desired line.
     (x2,y2) gives the ending coordinates of the line.

Current plot position on return: (x2,y2)

Function **mfprtf**

     This routine is an implementation of the standard **fprintf()** that writes its output
to the current plot frame using the current font. Extensions in **rkprintf()** are also
supported. This is the only way to combine textual and numeric information in a single
plotted string when using a font other than the built-in font, as the plot library
cannot compute the width of the text in variable-width fonts. The output text is
limited to a maximum of 255 characters.

Usage: int **mfprtf**(float x, float y, float ht, float angle, const char *fmt, ...)

Arguments: 'x' and 'y' give the position of the lower left corner of the plotted string.
     'ht' gives the height of the lettering. 'angle' gives the angle of the baseline in
     degrees above horizontal. 'fmt' is the standard **printf()**-style format description
     of the lettering, followed by arguments as required by the format string.

Return value: Number of characters written.

Current plot position on return: Undefined.

Functions **newplt** and **newpltw**

     These functions begin a new graph and specify initialization parameters. **newplt()**
begins a new graph in the initial (default) window if output is to a display device.
newpltw() begins a new plot in a window previously opened with **newwin()**. If output is
directed to a hard-copy device, the implementation will attempt to plot extra windows on
separate sheets or separate portions of roll paper, but this may not be possible. One
of these routines must be called once for each graph, before any other plotting calls
(except **factor()** or **setmf()** for setting global controls). On parallel computers, these
functions must be called in parallel from all nodes. Inbound followed by outbound
synchronization is performed.

Usage:
     int **newplt**(float xsiz, float ysiz, float xorg, float yorg, const char *pentyp,
        const char *pencol, const char *chart, int kout)
     int **newpltw**(float xsiz, float ysiz, float xorg, float yorg, const char *pentyp,
        const char *pencol, const char *chart, int kout, unsigned int iwin, unsigned
        int fmode)

Arguments: 'xsiz' is the x direction size of the total graph including all labels, etc.
     On devices that maintain an absolute scale, the coordinates are taken to be in
     inches. On other devices, the coordinates may be given in arbitrary units; the
     graph will be scaled such that a rectangle of size (xsiz,ysiz) just fits the
     available plotting area (maintaining the aspect ratio).

          'ysiz' is the y direction size of the total graph including all labels, etc.
     See discussion of 'xsiz'.

          'xorg' and 'yorg' define the plot origin. These values are added to all
     plotting coordinates in all frames prior to the application of any scale factors
     or frame coordinate transformations.

          'pentyp' specifies the type of pen to be used (e.g. "FELT-TIP", max 15 char).
     The allowed values are device-dependent (obtain them from user input). On visual
     display devices, this parameter is often used to indicate a line type (e.g.
     "SOLID", "DASHED", "DOTTED", etc.) A NULL pointer indicates that the default pen
     type is to be used.

          'pencol' is the same as in the **pencol()** call (max 15 char). A NULL pointer
     indicates that the default color ("BLACK") should be used.

          'chart' (max 32 chars) On hard-copy devices, this argument specifies the
     kind of chart paper to be used (e.g. "ALLWHITE"). On visual display devices, if
     this argument is anything other than the word "DEFAULT", the string is used as a
     title for the new plot. A NULL pointer can be used to indicate the default chart.

'kout' specifies which plot outputs are to be produced. This is intended to
allow only a subset of online graphs to be written to a metafile. The value of
'kout' should be the logical OR of whichever of the following named constants
(defined in plotdefs.h) are desired: SKIP_META causes writing of the metafile to
be skipped for this plot, even if metafile writing has been enabled by setmf().
SKIP_XG causes online X graphics to be omitted. DO_MVEXP causes this frame to be
exposed by a movie recording device if one is attached to the system.

'iwin' specifies the number of the window in which the new drawing is to be
placed (**newpltw()** only).

'fmode' specifies the frame mode for the initial frame on this plot
(**newpltw()** only; see description listed with **frmdef()** command).

Current plot position on return: (0,0). These coordinates will refer to the absolute
    position specified by the (xorg,yorg) arguments for subsequent plotting.

Value returned: 0 is returned for normal completion. BUT_INTX (defined as 1 in
    plotdefs.h) is returned if the user activated a device (keypress or mouse click)
    defined by the implementation as indicating that the application should be inter-
    rupted, for example, to read new input from its terminal. With mfdraw, this is
    the "INTERRUPT" button in the window menu. BUT_QUIT (defined as 2 in plotdefs.h)
    is returned if the user activated a device defined by the implementation as
    indicating that the application should be terminated.

Function **newwin**

This function declares the existence of a new plotting window and establishes
buffering for commands directed to that window. Plotting in this window does not begin
until a **newpltw()** call specifying the window number returned by **newwin()**. On parallel
computers, **newwin()** must be called in parallel from all nodes. Inbound followed by
outbound synchronization is performed.

Usage: unsigned int **newwin**(void)

Arguments: None.

Value returned: The number of the newly created window is returned. If **newwin()** is
    called before any call to **newplt()** or **newpltw()**, window 0 will be redundantly
    created. The window number may be used in future calls to **newpltw(), chgwin(),** or
    clswin().

Restriction: At most 65536 windows may be created in any one run.

Subroutines **number, nomber, numbc,** and **nombc**

These calls are used to plot floating point values as decimal strings (i.e. as
labels, not as points). **nomber()** and **nombc()**, respectively, are the same as **number()**
and **numbc()** except the order of the arguments has been changed to avoid a gap on the
stack and the name **nomber** is less likely to be found by **grep** in comments.

Usage:
     void **number**(float x, float y, float ht, double val, float angle, int ndec)
     void **nomber**(float x, float y, float ht, float angle, double val, int ndec)
     void **numbc**(float ht, double val, float angle, int ndec)
     void **nombc**(float ht, float angle, double val, int ndec)

Arguments: 'x', 'y' are the coordinates of the lower left-hand corner (before rotation)
    of the first character to be plotted. The pen is up while moving to this point.

'ht' is the character height. If negative or zero, height and angle from the
    previous call are used (enter the angle as '0'). If negative with **number()** or
    nomber(), the current string is concatenated to the end of the previous string
    (spacing may be incorrect if a font other than the default font is in use) and the

'x' and 'y' arguments are ignored. **numbc()** and **nombc()** provide this functionality without the need to enter negative 'ht' and meaningless 'x' and 'y' values.

'angle' is the angle at which the text will be plotted, in degrees, measured counter-clockwise up from the horizontal.

'val' is the floating point variable to be converted and plotted.

'ndec' is the number of places after the decimal to be plotted. If ndec = 0, the integer portion of the number is plotted with a decimal point; if ndec = -1, it is plotted without a decimal; if ndec < -1, the number is scaled by 10**abs(ndec+1) and rounded before plotting.

Current plotting position on return: Stored as the location at the lower right of the plotted characters assuming the default font was used, allowing following string concatenation calls to work properly on serial or parallel computers. Use **mfprtf()** to concatenate information plotted in other fonts.

Subroutine **pencol**

This routine changes the current plotting color to a specified name or explicit red-green-blue value. The implementation will select the nearest available color to the color requested. Plotting from all previous **plot()** calls is completed before the new 'pencol' argument takes effect. Subroutine **color()** may be used to request a change to a particular pen or color index; **rcolor()** may be used to request a change to a numeric color index returned by a previous call to **regcol()**.

Usage: void **pencol**(const char *pencol)

Argument: 'pencol' is the same as in the **regcol()** call (max 15 char). A NULL pointer causes the drawing color to be changed to "BLACK".

Current plotting position on return: unchanged

Parallel computation: On a parallel computer, each node can have its own current drawing color. Color changes affect only the node where executed.

Note: The limitation to 15 characters does not apply to registered color names.

Subroutine **penset**

This call requests a pen type or color change while a graph is being plotted. Plotting from all previous **plot()** calls is completed before the new 'pentyp' and 'pencol' arguments take effect. Keep in mind that pen changes may not be supported on some devices; operator intervention may be required with others. Note that **newplt()** and **newpltw()** implicitly call **penset()**--after each such call the pen type and pencol arguments given there will be in effect.

Usage: void **penset**(const char *pentyp, const char *pencol)

Arguments: 'pentyp' and 'pencol' are as in the **newplt()** call (max 15 char each).

Current plotting position on return: unchanged

Parallel computation: On a parallel computer, each node can have its own current pen type and color. Type and color changes affect only the node where executed.

Subroutine **pentyp**

This routine changes the type of pen used on a pen plotter and may change the style of line drawing (e.g. DASHED, DOTTED) on a visual display device depending on the implementation. Pen changes may not be supported on some devices; operator intervention may be required with others.

Usage: void **pentyp**(const char *pentyp)

Argument: 'pentyp' is the same as in the **newplt()** call (max 15 characters). A NULL
      pointer causes the default pen to be used.

Current plotting position on return: unchanged

Parallel computation: On a parallel computer, each node can have its own current pen
      type (not advisable where separate physical pens are involved). Pen changes
      affect only the node where executed.

Note: The limitation to 16 characters does not apply to registered pen types.

Subroutine **plot**

      This routine moves the figurative CRT beam or pen to a new location. The third
argument indicates whether or not a line is to be drawn from the current position to the
new position and whether or not the new position is to become the origin for subsequent
plotting. Any line that is drawn is drawn once only, regardless of the current retrace
setting.

Usage: void **plot**(float x, float y, int ipen)

Arguments: (x,y) give the coordinates of the point to which the pen is to be moved
      relative to the current plot origin.

          'ipen' controls drawing vs moving:
      ipen = PENDOWN (2) beam is on (pen is down) while moving, thus drawing a line.
      ipen = PENUP (3) beam is off (pen is up) during movement.
      ipen = -2 or -3 is same as +2 or +3, respectively, except the final location of
          the beam or pen becomes the origin from which coordinates are measured in
          subsequent plot library calls.

Note: Origin changes affect only the current plotting frame.

Current plotting position on return: (x,y) if ipen > 0, otherwise (0,0)

Parallel computation: Origin changes affect only the processor node where the call is
      made.

Subroutine **plotvers**

      This routine returns a text string containing the subversion repository revision
number of the current compilation of the plot library. It is recommended that this value
be printed in the output of the application in order to document exactly what version of
the plot library was used for each run.

Usage: char ***plotvers**(void)

Subroutine **polygn**

      This routine draws a regular closed polygon (or circle) in the current color.
Each edge of the polygon may be a straight line or it may be a "dented" line consisting
of two equal half-segments with the center displaced inward or outward to make a "star".
In addition, each vertex may be decorated with a "spike" consisting of a line segment
pointing some distance radially away from the vertex. These options permit a wide
variety of symmetric marker symbols to be produced.

Usage: void **polygn**(float xc, float yc, float radius, int nv, float angle, float dent,
      float spike, int kf, int kc)

Arguments: (xc,yc) give the location of the center of the polygon.

'radius' is the radius (distance from center to each vertex) of the desired
polygon.

'nv' is the number of vertices in the polygon. A value of '0' draws a
circle, a value of '2' draws a line, '3' a triangle, etc. Negative values and '1'
are invalid and cause termination of the calling program.

'angle' is an angle through which the polygon is rotated, measured in degrees
counterclockwise from a position in which one vertex is positioned along the
positive x axis.

'dent' is a factor by which 'radius' should be multiplied to obtain the
radius at the center of each edge of the polygon. A zero value indicates that the
edges should be straight lines. A value of 1.0 effectively causes a polygon with
twice the number of vertices to be drawn, but with spikes only on every other
vertex.

'spike' is a factor by which 'radius' should be multiplied to obtain the
radius of the end a spike line drawn radially outward (inward if spike < 1.0) from
each vertex of the polygon. A zero value indicates that no spikes should be
drawn.

'kf' is a fill control switch. If kf = -1 (FILLED), a filled polygon is
drawn. An unfilled polygon is drawn in the current retrace thickness if kf >= 0
(THIN or NORETRACE).

'kc' is a connectivity control switch, useful for making connected graphs.
If 'kc' = 2 (PENDOWN), a line is drawn from the current plotting position to the
center of the polygon. If 'kc' = 3 (PENUP) (or any other value), the connecting
line is omitted.

Current plot position on return: (xc,yc)

Subroutine **polyln**

This routine draws an open or closed polyline (a series of points connected by
lines) in the current color.

Usage: void **polyln**(int kf, int np, const float *x, const float *y)

Arguments: 'kf' is a fill control switch with values: 0 (THIN) an open, thin polyline
    is drawn; 1 (THICK) open and thick; 2 (CLOSED_THIN) closed and thin; 3
    (CLOSED_THICK) closed and thick; -1 (FILLED) closed and filled with color. In
    cases CLOSED_THIN, CLOSED_THICK, and FILLED, the first and last points are
    connected by a line to form a closed figure. In case FILLED, the polyline must be
    non-self-intersecting. In cases THIN and THICK, the first and last points are not
    joined (unless np = 2).

'np' is the number of points in the polyline.

'x' is an array of dimension 'np' containing the x coordinates of the points
    to be plotted.

'y' is an array of dimension 'np' containing the y coordinates of the points
    to be plotted.

Current plot position on return: x[np-1],y[np-1] if kf = THIN or THICK, otherwise,
    x[0],y[0]

Subroutine **rcolor**

Subroutine **rcolor()** changes the drawing color to a value returned by a previous
call to **regcol()** or to a preregistered color. The **regcol()**,**rcolor**() combination permits
the work of converting an English color name or hexadecimal color specification to a
color index to be performed just once, when **regcol()** is processed. Subsequent calls to

rcolor() are processed quickly and the space taken in the metafile is less than with
pencol(). Using a pen plotter that is capable of changing pens under program control,
an argument in the preregistered range will be interpreted as a pen number. (Keep in
mind that pen changes may not be supported on some plotters; operator intervention may
be required with others.)

Usage: void **rcolor**(unsigned int jc)

Argument: 'jc' must be a value returned by an earlier call to **regcol()**.

Current plot position on return: unchanged

Parallel computation: On a parallel computer, each node can have its own current
    drawing color. Color changes affect only the node where executed.

Subroutine **rect**

    This routine draws a rectangular box using the current color.

Usage: void **rect**(float x, float y, float wd, float ht, int kf)

Arguments: (x,y) give the location of the lower left-hand corner of the rectangle.

        'wd' is the width of the desired rectangle.

        'ht' is the height of the desired rectangle.

        'kf' is a fill control switch. If kf = -1 (FILLED), a filled rectangle is
    drawn. If kf >= 0, the rectangle is drawn using the current retrace thickness.

Current plot position on return: (x,y)

Function **regcol**

    Function **regcol()** assigns an integer value to an explicit color specifier. This
value may be used in subsequent calls to **rcolor()** to change the drawing color to the
regcol() argument. This method allows one to change calls to **pencol()** or **penset()** into
calls to **rcolor()** for more efficient performance (and shorter metafiles).

Usage: unsigned int **regcol**(const char *pencol)

Argument: 'pencol' specifies the color name to be registered. Only the first 15
    characters are significant. The allowed values are device-dependent and should
    generally be read from user input. The color may be a literal string (e.g.
    "RED"--see Appendix 1 for a list of color names that will be accepted by all
    implementations), or a 3-digit hexadecimal specification of blue, green, red
    values preceded by a letter 'X' (e.g. "XFF0" would specify cyan), or a 6-digit
    hexadecimal specification (2 digits blue, 2 digits green, 2 digits red) preceded
    by a letter 'Z' (used to obtain 24-bit color values).

Return value: Registration number for this color.

Note: **regcol()** merely assigns an integer synonym to a color name. The actual conver-
    sion to a color value is not performed until the plot is rendered on a device.
    Thus, the **regcol()**-**rcolor()** combination will yield exactly the same colors as
    calls to **pencol()** with the same arguments.

Current plot position on return: unchanged

Parallel computation: On a parallel computer, **regcol()** calls made on node 0 return
    values that may be copied to other nodes and used in **rcolor()** calls, where they
    will specify the same color. **regcol()** calls made on other nodes return values
    that are valid only on the node where registered.

Function **regfont**

      Function **regfont()** assigns an integer value to a font name. This value may be
used in subsequent calls to **rfont()** to change the font to the **regfont()** argument. This
method allows one to change calls to **font()** into calls to **rfont()** for more efficient
performance (and shorter metafiles).

Usage: unsigned int **regfont**(const char *font)

Argument: 'font' specifies the font name to be registered. Only the first 40
    characters are significant. The allowed values are device-dependent and should
    generally be read from user input.

Return value: Registration number for this font.

Note: **regfont()** merely assigns an integer synonym to a font name. The actual change to
    lettering with a new font is not performed until the plot is rendered on a device.
    Thus, the **regfont()**-**rfont()** combination will yield exactly the same font as calls
    to **font()** with the same arguments.

Current plot position on return: unchanged

Parallel computation: In a parallel computer, **regfont()** calls made on node 0 return
    values that may be copied to other nodes and used in **rfont()** calls, where they
    will specify the same font. **regfont()** calls made on other nodes return values
    that are valid only on the node where registered.

Function **regpen**

      Function **regpen()** assigns an integer value to a pen (drawing style) name. This
value may be used in subsequent calls to **rpen()** to change the drawing style to the
regpen() argument. This method allows one to change calls to **pentyp()** into calls to
rpen() for more efficient performance (and shorter metafiles).

Usage: unsigned int **regpen**(const char *pentyp)

Argument: 'pentyp' specifies the pen name (drawing style) to be registered. Only the
    first 15 characters are significant. The allowed values are device-dependent and
    should generally be read from user input.

Return value: Registration number for this pen type.

Note: **regpen()** merely assigns an integer synonym to a pen name. The actual change to
    drawing with a new pen type is not performed until the plot is rendered on a
    device. Thus, the **regpen()**-**rpen()** combination will yield exactly the same drawing
    style (pen on a hard-copy plotter) as calls to **pentyp()** with the same arguments.

Current plot position on return: unchanged

Parallel computation: In a parallel computer, **regpen()** calls made on node 0 return
    values that may be copied to other nodes and used in **rpen()** calls, where they will
    specify the same pen or drawing style. **regpen()** calls made on other nodes return
    values that are valid only on the node where registered.

Subroutine **retrace**

      Subroutine **retrace()** changes the thickness of all subsequent lines (including
arcs, ellipses, circles, rectangles, squares, and symbols drawn with stroke fonts). It
does not affect solid objects (rectangles, circles, ellipses), bitmaps or bitmap fonts,
or lines drawn by calling **plot()**.

Usage: void **retrace**(int krt)

Argument: 'krt' is the retrace parameter. If krt = 0 (THIN or NORETRACE), subsequent
    lines are drawn once only. If krt > 0 (THICK or RETRACE), subsequent lines are

retraced for greater density. In some implementations, the numeric value of krt
(0 <= krt <= 15) may control the thickness of subsequent lines in approximately
0.01 inch intervals.

Current plotting position on return: unchanged

Parallel computation: On a parallel computer, each node can have its own current line
thickness. Retrace changes affect only the node where executed.

Subroutine **rfont**

Subroutine **rfont()** changes the lettering font to a value returned by a previous
call to **regfont()**. This combination permits the work of converting a font name to a
font index to be performed just once, when **regfont()** is processed. Subsequent calls to
rfont() are processed quickly and the space taken in the metafile is less than with
font().

Usage: void **rfont**(unsigned int jf)

Argument: 'jf' must be a value returned by an earlier call to **regfont()**.

Current plot position on return: unchanged

Parallel computation: On a parallel computer, each node can have its own current font.
Font changes affect only the node where executed.

Subroutine **rpen**

Subroutine **rpen()** changes the pen (drawing style) to a value returned by a previ-
ous call to **regpen()**. This combination permits the work of converting a drawing style
name to a pen index to be performed just once, when **regpen()** is processed. Subsequent
calls to **rpen()** are processed quickly and the space taken in the metafile is less than
with **pentype()**.

Usage: void **rpen**(unsigned int jp)

Argument: 'jp' must be a value returned by an earlier call to **regpen()**.

Current plot position on return: unchanged

Parallel computation: On a parallel computer, each node can have its own pen type. Pen
type changes affect only the node where executed. However, with an actual pen
plotter, frequent changes are likely to be very inefficient.

Subroutine **setmf**

Subroutine **setmf()** initializes the plot library for metafile graphics. Metafile
graphics may be plotted immediately if an X server is available, and may optionally be
stored in a real metafile. The two functions are independent--metafile output may be
created and sent to a file alone, an X windows server alone, or to both devices. If
used, **setmf()** should be called before the first **newplt()** call in the run.

Usage: void **setmf**(const char *fname, const char *station, const char *title,
const char *icon, long buflen, ui32 dbgmask, int enc, int lci, int lcf)

Arguments: 'fname' is the name of the metafile to be written. If 'fname' is a null
pointer, no file is written. If 'fname' is a pointer to a null string or the
string "-", output is written to standard output.

'station' is the name of a graphics station where X windows output should be
displayed, e.g. "acteon:0.0". A null pointer indicates that no X output should be
generated. A pointer to a null string indicates that the display specified by the
environment variable 'DISPLAY' should be used, and if that is not present, then
the login host should be used.

          'title' is a text string to be displayed in the window title bar for the
graphics output. It should generally be the name of the application program. A
null pointer or a pointer to a null string causes the title to be left blank.

          'icon' is a pathname specifying a file containing an icon image to be used
when the X graphics window is minimized. A null pointer or a pointer to a null
string indicates that a generic icon should be used.

          'buflen' is the length of the internal buffer to be allocated for metafile
data. This should normally be at least equal to the hardware blocksize on the
device where the metafile is written. If 'buflen' is less than the default value
(currently 4096 bytes), the default is used.

          'dbgmask' contains undefined data which may be used during debugging. It is
OR'd bitwise with any debugging mask passed from the driver program in a parallel
computer and transmitted to mfdraw. Currently, the bit setting '0xc0000000' will
cause mfdraw to enter a "spin-wait" loop, waiting for a debugger to be attached.

          'enc' indicates the metafile encoding to be used. 'B' indicates binary.
Other values are obsolete or reserved for future use. and generate an error with
the current call.

          'lci' is the length of the integer part of coordinate values in bits. '0'
indicates that the default should be used, currently 6. Values less than the
default will be set to the default.

          'lcf' is the length of coordinate fractions in bits. '0' indicates that the
default should be used, currently 10. The total of 'lci' + 'lcf' may not exceed
28. If this total is exceeded, 'lcf' will be reduced to 28 - lci.

Parallel computation: On parallel computers, **setmf()** only needs to be called from node
     0. If called on other nodes, no error occurs, but the call is ignored.

Subroutine **setmovie**

          Subroutine **setmovie()** sets the plot library to display plots as fast as they are
computed (batch or movie mode) or to hold each plot on the display until the operator
takes some implementation-dependent action, such as typing a particular key or pressing
a mouse button (still mode). In batch mode, the X window data stream and metafile are
closed at once when **endplt()** is called and the application continues, presumably to
termination. This usually causes the last plot to disappear. This mode permits unat-
tended generation of movies from script files. Movie mode is the same as batch mode on
systems that leave the last plot on the display device. Otherwise, movie mode pauses
when **endplt()** is called until the user gives permission for termination. This mode
assures that the last plot (perhaps generated during an overnight run) can be viewed by
the user before termination. On X-windows systems, a dialog box appears with two
choices: QUIT causes the application to terminate, while CONTINUE simply deletes the
dialog box so the last plot can be viewed. QUIT must then be chosen from the background
menu to complete termination.

          In addition, **setmovie()** sets the plot library to generate output on an attached
film recorder, or to activate the beeper on the login terminal when each plot is com-
plete and **newplt()** was called with domvexp TRUE (this may be used to expose one frame of
film on a movie camera placed in front of the terminal).

          When called before the first **newplt()** or **newpltw()** call, **setmovie()** affects the
default first window. After one of those calls, it affects the current window.

Usage: void **setmovie**(int mmode, int device, int nexpose)

Arguments: 'mmode' specifies the desired movie mode. A value of MM_NOCHG (0) leaves
     the previous setting unchanged. A value of MM_STILL (1) changes to still mode. A
     value of MM_MOVIE (2) changes to movie mode. A value of MM_BATCH (3) changes to
     batch mode. The initial default is still mode. Use of MM_NOCHNG permits an

application to change 'nexpose' without knowing the current movie mode, which may
have been modified as a result of user interaction with an X window server.

'device' specifies the desired movie output device. A value of MD_MATRIX (0)
indicates that a MATRIX film plotter is to be used. A value of MD_BEEPER (1)
indicates that the NSI electrically driven movie camera is to be used. Other
values may be defined locally.

'nexpose' specifies the number of exposure to be made of each plot.

Current plotting position upon return: unchanged

Parallel computation: On parallel computers, this routine only needs to be called from
    node 0. If called on other nodes, no error occurs, but the call is ignored.

Subroutine **square**

This routine draws a square using the current color.

Usage: void **square**(float x, float y, float edge, int kf)

Arguments: (x,y) give the location of the lower left-hand corner of the rectangle.

'edge' is the edge width of the desired square.

'kf' is a fill control switch. If kf = -1 (FILLED), a filled square is
drawn. If kf >= 0, the square is drawn using the current retrace thickness.

Current plot position on return: (x,y)

Subroutines **symbol**, **simbol**, **symbc**, and **simbc**

These calls are used to plot labels or other alphanumeric information. **symbol()**
plots the text at a specified (x,y) position; **symbc()** plots it at the current drawing
position. If **font()** has been called, and the specified font is available, that font is
used. (If fonts are not scalable, the nearest size to the specified letter height is
chosen.) Otherwise, lettering is drawn as individual strokes using an internal font.
Use of the default font assures that the aspect ratio and appearance of the lettering is
the same on all supported devices. **simbol()** and **simbc()**, respectively, are the same as
symbol() and **symbc()** except the order of the arguments has been changed to avoid a gap
on the stack and the name **simbol** is less likely to be found by **grep** in comments.

Usage:
```
    void symbol(float x, float y, float ht, const char *text, float angle, int n)
    void simbol(float x, float y, float ht, float angle, const char *text, int n)
    void symbc(float ht, const char *text, float angle, int n)
    void simbc(float ht, float angle, const char *text, int n)
```

Arguments: 'x', 'y' are the coordinates of the lower left-hand corner (before rotation)
    of the first character to be plotted. The pen is up while moving to this point.

'ht' is the character height. If negative or zero, height and angle from the
previous call are used. If negative, the current string is concatenated to the
end of the previous string. When negative values of 'ht' are used with **symbol()**,
the 'x' and 'y' arguments are ignored.
    With the default font, characters are plotted on a 7 x 4 grid with spacing of
an additional 2 units to the right of each character (i.e. total character width
is 6/7 times 'ht'). Best results will be obtained if the height is a multiple of
7 times the pixel size or plotter increment size (usually 0.01 inches).

'text' is the label to be plotted. It may be given as a literal or as a
pointer to an array containing a character string. Special characters (such as
Greek letters) are available and are listed in Appendix 2. These characters are
not necessarily available in all fonts. Note that for historical reasons, the

length of the string must be encoded in the 'n' parameter, however, text will be
truncated at the normal C end-of-string indication if shorter than implied by 'n'.

'angle' is the angle at which the text will be plotted, in degrees, measured
counter-clockwise up from the horizontal.

'n', if positive, is the number of characters drawn from the string defined
by 'text'. If n <= 0, a single character is drawn, centered on (x,y). If n = -1
or 0, the pen is up while moving to the character position; if n < -1, it is down.
Centered characters are intended for use in making line graphs with markers.

Current plotting position on return: Stored as the location at the lower right of the
plotted characters assuming the default font was used, allowing following string
concatenation calls to work properly on serial or parallel computers. Use
mfprtf() to concatenate information plotted in other fonts.

Notes: Characters are assumed to be coded in the native character set of the host
system, i.e. EBCDIC for IBM370 and ASCII for all others. All character strings
are converted to ASCII for encoding in metafiles. It would be nice if the current
drawing location on return from a **symbol()** or **number()** call could be set to the
location where the next letter should go to continue the text string, but it is
not clear that this can be accomplished in general when proportional-spaced fonts
are used, so this feature is not included in the present specification.

Function **updwin**

This function may be called to indicate to the plot library that all drawing com-
mands relating to the current plot in the specified window should be drawn. This allows
a plot to be updated with partial information or finished on an online display even if
newplt(), **newpltw()**, or **endplt()** is not called immediately. User interface buttons are
not returned.

Usage: void **updwin**(unsigned int iwin)

Argument: 'iwin' is the number of the window whose plot is to be updated. '0' is valid
and refers to the default window.

Parallel computation: Call from all nodes. This provides synchronization of plot
output.

Subroutine **where**

This routine returns the current plotting position and scale factor. These values
are not affected by any change the user may make by dragging or resizing a local drawing
frame. Keeping in mind that the current plotting position is undefined when not speci-
fied in the individual subroutine description, **where()** may be used to retrieve the plot-
ting position in these cases or when the position is the result of a computation, as
with the **simbol()** and **number()** routines.

Usage: void **where**(float *x, float *y, float *fac)

Arguments: 'x', 'y' are variables to receive the last coordinates.

'fac' is a variable to receive the last scale factor.

Current plotting position on return: unchanged

Parallel computation: The drawing location on the node executing the call is returned.

APPENDIX 1: COLOR NAMES ALWAYS ACCEPTED BY **penset()** AND **pencol()**

     The following color names will be recognized by all implementations of **penset()**
and **pencol()**. However, if the device does not support the specified color, a similar
color may be chosen by the implementation. Implementations are free to support
additional color names not listed here.

          WHITE, BLACK, BLUE, CYAN, MAGENTA, VIOLET, ORANGE, GREEN, YELLOW, RED.

White and black are defined with respect to a pen plotter, i.e. white is the background
color and black is the darkest drawing color. On a visual display device, black will
generally be mapped to white and white to black.

APPENDIX 2: SPECIAL CHARACTERS AVAILABLE WITH SYMBOL ROUTINE

     The following characters are supported by all implementations of **symbol()** at the
ASCII code points given. There is currently no provision to use Unicode. Characters
marked with an asterisk (*) are specifically designed for centered plotting, i.e. for
use as graph markers. These characters are mapped to the same positions in the ASCII
and EBCDIC implementations so that programs may invoke them using their integer values.
Characters marked with two asterisks (**) are not plotted, but rather cause the current
plotting position (carriage return, backspace) or character size and plotting position
(subscript, superscript) to be changed. The up, down, left, and right arrow characters
are mapped to their positions in the IBM PC character set. Greek letters and other
special characters are mapped to positions following character 0x7f in the ASCII set and
before 0x7f in the EBCDIC set. Unless otherwise stated, Greek letters are lower case.
Characters not defined in the official ASCII or EBCDIC codes are mapped arbitrarily.

Dec Code	Hex Code	ASCII character	EBCDIC character
0	00	Centered square*	Centered square*
1	01	Centered octagon*	Centered octagon*
2	02	Centered triangle*	Centered triangle*
3	03	Centered plus*	Centered plus*
4	04	Centered X*	Centered X*
5	05	Centered diamond*	Centered diamond*
6	06	Small centered square*	Small centered square*
7	07	Centered square with rays at corners*	Centered square with rays at corners*
8	08	Backspace**	Backspace**
9	09	Horizontal Tab**	Horizontal Tab**
10	0a	New Line**	New Line**
11	0b	Vertical Tab**	Vertical Tab**
12	0c	Form Feed**	Form Feed**
13	0d	Carriage return**	Carriage return**
14	0e	Star	Star
15	0f	Horizontal line	Horizontal line
16	10	Vertical line	Vertical line
17	11	And (inverted v)	And (inverted v)
18	12	Superscript**	Superscript**
19	13	Equivalence	Equivalence
20	14	Subscript**	Subscript**
21	15		
22	16	Not equals	Not equals
23	17	Plus-minus	Plus-minus
24	18	Up arrow	Up arrow
25	19	Down arrow	Down arrow
26	1a	Right arrow	Right arrow
27	1b	Left arrow	Left arrow
28	1c	Implication	Implication
29	1d	Inclusion	Inclusion
30	1e	Squiggle	Squiggle
31	1f	Double squiggle	Double squiggle
32	20	Blank	Blank
33	21	Exclamation point	
34	22	Quotation marks	Mu
35	23	Pound sign	Pi

PLOTTING ROUTINES

36	24	Dollar sign	Phi
37	25	Percent	Theta
38	26	Ampersand	Psi
39	27	Apostrophe	Chi
40	28	Left parenthesis	Omega
41	29	Right parenthesis	Lambda
42	2a	Asterisk	Alpha
43	2b	Plus	Delta
44	2c	Comma	Epsilon
45	2d	Minus	Eta
46	2e	Period	
47	2f	Slash	
48	30	0	Summation
49	31	1	Division
50	32	2	Less than or equals
51	33	3	Greater or equals
52	34	4	Double underscore
53	35	5	Overscore
54	36	6	
55	37	7	Upper case delta
56	38	8	Gamma
57	39	9	Square root
58	3a	Colon	Double dagger
59	3b	Semicolon	Triple dagger
60	3c	Less than	Caret
61	3d	Equals	Times
62	3e	Greater than	Infinity
63	3f	Question mark	Em Dash
64	40	At	Blank
65	41	A	
66	42	B	
67	43	C	
68	44	D	
69	45	E	
70	46	F	
71	47	G	
72	48	H	
73	49	I	
74	4a	J	Cents
75	4b	K	Period
76	4c	L	Less than
77	4d	M	Left parenthesis
78	4e	N	Plus
79	4f	O	Vertical bar
80	50	P	Ampersand
81	51	Q	
82	52	R	
83	53	S	
84	54	T	
85	55	U	
86	56	V	
87	57	W	
88	58	X	
89	59	Y	
90	5a	Z	Exclamation point
91	5b	Left bracket	Dollar sign
92	5c	Backslash	Asterisk
93	5d	Right bracket	Right parenthesis
94	5e	Caret	Semicolon
95	5f	Underscore	Not
96	60	Back apostrophe	Minus
97	61	a	Slash
98	62	b	
99	63	c	
100	64	d	
101	65	e	
102	66	f	

PLOTTING ROUTINES

103	67	g	
104	68	h	
105	69	i	
106	6a	j	Split vertical bar
107	6b	k	Comma
108	6c	l	Percent
109	6d	m	Underscore
110	6e	n	Greater than
111	6f	o	Question mark
112	70	p	
113	71	q	
114	72	r	
115	73	s	
116	74	t	
117	75	u	
118	76	v	
119	77	w	
120	78	x	
121	79	y	Back Apostrophe
122	7a	z	Colon
123	7b	Left brace	Pound sign
124	7c	Vertical bar	At
125	7d	Right brace	Apostrophe
126	7e	Tilde (= squiggle)	Equals
127	7f	Delete (ignored)	Quotation marks
128	80	Double underscore	
129	81	Overscore	a
130	82	Integral	b
131	83		c
132	84		d
133	85		e
134	86		f
135	87		g
136	88		h
137	89		i
138	8a	Centered Y*	Centered Y*
139	8b	Centered Z*	Centered Z*
140	8c	Centered asterisk*	Centered asterisk*
141	8d	Centered double bar X*	Centered double bar X*
142	8e	Centered vertical bar*	Centered vertical bar*
143	8f	Summation	
144	90	Division	
145	91	Less than or equals	j
146	92	Greater than or equals	k
147	93	Upper case delta	l
148	94	Gamma	m
149	95	Square root	n
150	96	Double dagger	o
151	97	Triple dagger	p
152	98	Times	q
153	99	Cents	r
154	9a	Not	
155	9b	Infinity	
156	9c	Degrees	
157	9d	Em Dash	
158	9e		
159	9f		
160	a0		
161	a1		Tilde (= squiggle)
162	a2		s
163	a3	Mu	t
164	a4	Pi	u
165	a5	Phi	v
166	a6	Theta	w
167	a7	Psi	x
168	a8	Chi	y
169	a9	Omega	z

PLOTTING ROUTINES


```
170          aa          Lambda
171          ab          Alpha
172          ac          Delta
173          ad          Epsilon                    Left bracket
174          ae          Eta

189          bd                                     Right bracket

191          bf                                     Em dash
192          c0                                     Left brace
193          c1                                     A
194          c2                                     B
195          c3                                     C
196          c4                                     D
197          c5                                     E
198          c6                                     F
199          c7                                     G
200          c8                                     H
201          c9                                     I

204          cc                                     Integral

208          d0                                     Right brace
209          d1                                     J
210          d2                                     K
211          d3                                     L
212          d4                                     M
213          d5                                     N
214          d6                                     O
215          d7                                     P
216          d8                                     Q
217          d9                                     R

224          e0                                     Backslash
225          e1
226          e2                                     S
227          e3                                     T
228          e4                                     U
229          e5                                     V
230          e6                                     W
231          e7                                     X
232          e8                                     Y
233          e9                                     Z

240          f0                                     0
241          f1                                     1
242          f2                                     2
243          f3                                     3
244          f4                                     4
245          f5                                     5
246          f6                                     6
247          f7                                     7
248          f8                                     8
249          f9                                     9
250          fa                                     Vertical bar
255          ff          Delete (ignored)           Delete (ignored)
```

<u>INDEX TO PLOTTING SUBROUTINE CALLS</u>

PLOTTING ROUTINES

- End -