Bellcore MGR Window Manager

Version 0.112
August, 1993

Applix 1616 microcomputer project
Applix pty ltd

Bellcore MGR Window Manager

Even though Applix has tested the software and reviewed the documentation, Applix makes no warranty or representation, either express or implied, with respect to software, its quality, performance, merchantability, or fitness for a particular purpose. As a result this software is sold "as is," and you the purchaser are assuming the entire risk as to its quality and performance.

In no event will Applix be liable for direct, indirect, special, incidental, or consequential damages resulting from any defect in the software or its documentation.

The Applix portions of this manual were written by Andrew Morton
Additional introductory and tutorial material by Eric Lindsay

Comments about this manual or the software it describes should be sent to:

Applix Pty Limited
324 King Georges Road
Beverly Hills 2209
N.S.W. Australia
(02) 758 2688

ISBN 0 947341 ?? ?

MC68000®™ is a trademark of Motorola Inc.

1
Bringing up MGR

Bringing up the Bellcore window manager MGR on the Applix 1616, by Andrew Morton, 28 Sept 1989.

To run mgr, a few ASSIGNments, files, and programs must first be set up, as detailed below.

ROM version

mgr is supported on 1616/OS version 4.1a and higher.If you haven't updated to at least this version, then it won't work.For best results, you also require more than a half megabyte of memory, although small tasks will run without memory expansion.

ASSIGNments

The path `/mgr` must be assigned to point to the directory under which the mgr font, icon and library directories must be found. Specifically, the following directories must be available:

`/mgr/icon`	Icons
`/mgr/font-16`	Fonts
/mgr/lib	Libraries & include files for mgr C language application development.

Environment

mgr and a number of its application programs expect to be able to read a file called `/usr/lib/environ`. Assignments must be set up so that this file is readable. Assign `/usr` or `/usr/lib`, rather than assigning all of `/usr/lib/environ`. This is because other applications expect to find files under `/usr/lib`.

The environment file consists of lines of the form:

```
THINGY=setting
```

with no white space around the '=' sign. Upper and lower case is significant in this file. The following environment settings must be defined in the environment file for mgr to work:

MAX_X
The width of the video display in pixels minus one.

MAX_Y
The height of the video display in pixels minus one.

WIDE
The width of the video display in pixels.

HIGH
The height of the video display in pixels.

HOME
This should be assigned to `/mgr`.

MGRSCREENINIT
Must be set to the name of a program which initialises the video display hardware for WIDE x HIGH pixels. mgr must be able to EXEC this program. This can be a multiple assignment, such as

```
MGRSCREENINIT=serial a .1200 8 8 0 0!GO_1024
```

MGRCLEANUP
Set to the name of a program which mgr EXECs before exitting: can be used for restoring the video mode. Other optional settings in the environment file are:

MICROSOFTMOUSE
Normally mgr expects a three button 5 byte packet PC-systems mouse. If MICROSOFTMOUSE is defined then the mouse driver expects a two button 3 byte packet microsoft mouse. mgr requires a three button mouse for some applications, so the microsoft mouse driver uses the combined press of both buttons to simulate the third (rightmost) button on the PC-systems mouse.

If you are using a microsoft mouse program set the serial port in the following manner:

```
serial a .1200 7 7 0 0
```

The PC-systems mice use the following settings:

```
serial a .1200 8 8 0 0
```

MOUSEXSCALE, MOUSEYSCALE
Multipliers for mouse movements. If not set these default to 100, which is a 1:1 multiplier. Increase these settings to increase the mouse sensitivity.

NONLINEARMOUSE
If this is present in the environment file, mgr uses a non-linear scaling algorithm which moves the mouse further when it is moving faster, which takes some getting used to, but reduces large mouse motions, whilst permitting accurate work. The MOUSEXSCALE and MOUSEYSCALE settings still take effect when non-linear mode is selected.

JOYSTICK
If present in the file, this causes mgr to use the joystick for mouse positioning. Doesn’t work very well.

DEBUG
If present in the file, this setting causes most of the mgr application programs to produce debug output on their standard error.

The program `setenv.xrel` is supplied for altering the environment file. Usage is

setenv [THINGY]
setenv [THINGY=setting]

The first mode removes the definition of THINGY from `/usr/lib/environ`. The second gives it a new setting.

Files

There are a number of other special files for mgr:

`/mgr/.mgrc`
The mgr startup file: see mgr.man for details.

`/mgr/font-16/.mgrc`
The mgr font startup file: see mgr.man for details.

2
Invoking mgr

Usage is

```
mgr [flags]
```

the flags are as follows:

-c
: Run the Bellcore copyright program on startup.

-dSS
: Set debug mode, using chars in SS as flags. As supplied mgr has been compiled with debugging disabled.

-x
: Do not run a startup file. See mgr.man.

-m filename
: Use the file ‘filename’ as the source of mouse input. Presumably sa: or sb:, mgr expects to see a 3 button 5 byte/packet mouse.

-s filename
: Get startup commands from ‘filename’ instead of from `/mgr/.mgrc` on startup.

-F filename
: Use ‘filename’ as the default mgr font, instead of the inbuilt one. Use a full pathname, such as
  ```
  mgr -F /mgr/font-16/oldeng.fnt
  ```

-f dirname
: Look in the specified directory for fonts, rather than `/mgr/font-16`.

-i dirname
: Look in the specified directory for icons, rather than `/mgr/icon`.

Running under mgr

Basically everything goes as per the mgr documentation. The system menu (activated by the left button) has been modified so that it displays the name of the character device associated with the window which the button is clicked on. This has been done so that the user can direct the output of a program to that character device and have it come out on the appropriate window.

The ‘buckey’ keys are obtained by holding down ‘ALT’ whilst typing a key. The ALT-N sequence has been altered to ALT-A.

If you are running 1616/OS V4.1a, use ALT-O in place of the usual ALT-S to suspend output in the current window.

3
Writing mgr applications

How to do it, from Andrew Morton, 28 Sept 1989

The mgr user manual comprehensively describes how to write programs which use mgr's features. The manual is quite complicated and very much written from the standpoint of the C language.

Because mgr's communications with client programs are performed entirely via standard input and standard output, mgr applications can be written in ANY language which has available a getchar function and a putchar one. In fact an mgr program can run on another machine which is not running mgr: all it need do is send up the appropriate escape sequences and read back and interpret the results.

The mgr documantation needs to be reworked in a more language independent manner, documenting the raw escape sequences, rather than the C macro libraries (unfortunately, this isn't real high on the priority list).

The basic format of an MGR escape sequence is

ESCC
ESCnnC
ESCnn,nnC
ESCnn,nn,C
etc.

where ESC is the escape character, 'nn' is a decimal number (a sequence of digits in the range '0' to '9') and C is a single character. The commas are commas.

If a variable length string is to be uploaded (such as with a menu upload) the length of the string is coded as a decimal number and then that number of characters are sent.

All of mgr's terminal emulation codes follow this scheme also. I have added a TVI925/applix 1616 terminal emulator front end to mgr which translates the normal 1616 escape sequences coming from the application programs into mgr sequences. Most of the 1616 escape codes are supported. mgr patches the DEF_WIND system call so that it can provide the correct window dimensions to those programs (specifically EDIT) which use DEF_WIND to read the current 1616 video driver window dimensions.

No attempt has been made to translate the 1616 graphics system calls into mgr escape sequences: the overhead of encoding and decoding the escape sequences greatly slow the graphics performance. mgr is not really a graphics program; it is an enhancement to the command line shell. Use the 1616 SSEG! video graphics package for serious graphics work.

Applix's Dr Doc word processor for the 1616 has been modified so as to use mgr's facilities for implementation of pull-down menus, scroll bars, point-and-click cursor positioning, block marking via the mouse, etc. The source code for the mgr part of Dr Doc is included on the Dr Doc distribution disk, along with the whole editor in linkable form. This is a good demonstration of how an mgr interface can be tacked onto an existing application.

4
Installation of mgr

Installation of the Bellcore window manager mgr in the Applix 1616, by Andrew Morton, 28 September, 1989

The window manager mgr was orginally designed to run on Sun workstations under SunOS, an enhanced implementation of Unix. The 1616 version retains practically all the functionality of the original. There are, in fact, a few 1616 specific enhancements.

Suns have large video monitors: 1152(W) pixels by 912(H) pixels. This sort of resolution is needed to run a windowing shell such as mgr. To get a reasonable resolution from the 1616's inbuilt video mgr requires that the 30MHz video circuitry modification be done. This modification involves the replacemant of an IC on the 1616 motherboard (U3: 74LS298) with a 20 pin 16L8B PAL. The PAL preserves the functions of the 298, as well as permitting the 30MHz 1 bit/pixel mode.

Installation

Remove U3. Put the PAL in U3's socket with pins 1, 2, 19 and 20 hanging out. Solder 3 pieces of wire onto the pins of U3 to make the following connections:

U3 pin 1 to U3 pin 12
U3 pin 2 to U27 (6845) pin 17
U3 pin 20 to +5.

The +5 connection is best done to the plate-through hole in the wide trace leading to the collector of Q1, between R9 and U5.

That's it! The connection to pin 1 is to permit the PAL to determine whether to display the border colours or the display colours. Pin 17 of the 6845 is an unused address line output; it is used here to select the video mode: setting the start address register (register 12) in the 6845 to $20 selects 30MHz mode; setting it to zero selects 15MHz mode.

Fixing horizontal sync

The HSYNC signal (pin 8 of the video connector) is in fact CSYNC in the 1616; that is, it is HSYNC ORed with VSYNC, rather the HSYNC on its own. This is a design flaw. Some monitors (especially when you push them) will exhibit tearing or complete loss of sync because of the loss of horizontal sync during vertical sync. It is therefore recommended that you perfrom the following modification, which brings the true horizontal sync signal out, without affecting the composite video signal.

Cut the track leading to pin 11 of U2.
Connect pin 11 of U2 to pin 39 of U27.

This modification significantly improves the appearance of Conal Walsh's EGA video mode software on multisync monitors, and probably makes even more difference on straight EGA and dual-scan monitors.

Inverting vertical sync

When mgr is used in high resolution modes such as 960x512 some 'smart' monitors will detect the new line rate and will decide that the vertical sync signal should be inverted. If you are experiencing problems with either a dual-sync or multi-sync monitor it is quite possible that vsync needs inversion.A simple way to invert vsync is to use the spare inverter at pin 1 of U54.

Cut the track going to U54 pin 1 on the solder side of the 1616 PCB and run a wire from U54 pin 1 to vsync (U27 pin 40).

Pull pin 8 of U2 out of its IC socket and connect it to inverted vsync (U54 pin 2).

Of course this scheme is not software switchable. It can be improved by taking both vsync (U27 pin 40) and inverted vsync (U54 pin 2, as above) to either side of a 3 pin strapping block.

Connect the middle pin of the strapping block to U2 pin 8, as above. The sync polarity can now be swapped using a shorting shunt on the 3 way strap.

The best way of performing the sync inversion is to make it software programmable by adding another IC to the board. An exclusive-OR gate does the job, but a single output bit needs to be found for doing the selection. The most significant bit of the border colour latch is an appropriate signal to use, as border colours brighter than 7 are unlikely to be used. Perform the following steps:

1) Adding the XOR gate

Put a 74LS86 exclusive-or gate in at the spare IC position U65. Align it with pin 1 of the 20 way socket, so pin 14 of the 74LS86 goes into pin 20 of the socket. Connect pin 7 of the 74LS86 to pin 10 of the socket.

2) The select bit

Remove U16 from its socket, bend out pin 6 and replace the chip. This pin is BC3, the most significant bit of the display border colour. Run a wire from the floating pin 6 to pin 1 of the new 74LS86.

3) Restoring the border colours

As bit 3 of the border colour is now used for something else we must do something about the border colours: connect the BC3 signal to ground by running a wire between pins 6 and 10 of U16's socket, on the solder side of the PCB.

4) Connecting up the XOR gate

Run a wire from pin 40 of U27 (vsync) to pin 2 of the 74LS86. Remove U2 from its socket, bend out pin 8 and replace U2 in the socket. Connect a wire from the floating pin 8 of U2 to pin 3 of the 74LS86.

Normally border colours in the range of 8-15 are not used, so BC3 will be low. This means that vsync is passed through the 74LS86 unchanged. To invert vsync you must add 8 to the current border colour. From assembly language:

```
move.l   #36,d0    * set_bdcol system call
clr.l    d1
move.b   $301,d1   * video latch shadow register
lsr.b    #4,d1     * Get border colour bits
and.b    #7,d1     * Preserve current setting
* Leave the next line out for normal sync
or.b     #8,d1     * Set BC3
trap     #7        * New colour
```

From the command line:

```
syscall .36 8
```

Programming

In 30MHz mode the pallette registers must have the following settings:

Pallette entry Setting

0	0
1	3
2	12
3	15

The whole object of the exercise is to persuade your monitor to produce the maximum number of pixels. This is inevitably done by experimentation. Look at the many different initialisation programs supplied, pick and/or modify one to you and your monitor’s taste.

5
mgr - manage windows on a SUN Workstation

Synopsis

```
mgr [ -ffont_dir ] [ -iicon_dir ] [ -sstartup_file ] [ -n ] [ -x ] [ -v ] [
        -V ] [ -Fdefault_font ] { [ -dlist ] [ -m mouse_device ] [ -Bwin-
        dow_buff ] [ -bshell_buff ] [ -Ppoll_interval ] [ -Sscreen ] }
```

Description

Mgr is a window manager for the SUN workstation, now ported to the Applix 1616 by Andrew Morton.

It permits the creation and manipulation of overlapping windows, with different processes running in each window. The user controls the function and layout of the display with a mouse. Windows are updated asynchronously even if they are partially (or completely) obscured by other windows, although obscured windows may arrange to have their output suspended until the window is uncovered.

Each window runs a terminal emulator which, in addition to the functions normally required to run screen oriented programs, such as vi, provides primitives for drawing lines, doing bit-blts, and performing administrative functions such as reshaping the window, changing fonts, or starting a new window. Details of of the terminal emulator operation are described in the MGR - C Language Application Interface.

The useful command line options are:

-ffont_dir
: Use font_dir as the directory to find the fonts, instead of `/usr/mgr/font`.

-iicon_dir
: Use icon_dir as the directory to find the icons, instead of `/usr/mgr/icon`.

-sstartup_file
: Use startup file instead of `$HOME/.mgrc` to obtain initial configuration information. See the description of startup commands below.

-n
: Bitmap files are created using the new, portable bitmap format. The portable format has an 8 byte header, and each row is padded to a byte boundary. Ordinarily the old (6 byte) bitmap header is produced, followed by the bitmap data with each line padded to an 16 bit boundary. Eventually, the sense of -n will change, when all of the programs that were written in the old format are changed.

-x
: Don’t use a startup file upon execution.

-v
: Don’t run MGR at all. Print the current version number and creation date instead.

-V Just like -v above only prints the compile flags used to make MGR and its home directory.

-Fdefault_font
Use default_font as the pathname of a MGR font to be used in place of MGR's builtin default font.

The rest of the options are:

-dlist
Print debugging information on stderr. list is one or more of the characters: *ABCEFMNPSUbcdefilmnopsuwxy each of which turns on debugging output for some aspect of MGR.

-mmouse_device
Use mouse_device instead of `/dev/mouse` to obtain mouse coordinates.

-Sscreen
Use screen instead of `/dev/bwtwo0` as the display device.

-Bwin_buff
Process characters to a window in up to win_buff byte chunks (the default is 40).

-bshell_buff
Buffer up to shell_buff bytes of output from a program before writing it on a window (the default is 256).

-Ppoll_interval
When output is pending in a window, wait poll_interval micro-seconds on every polling loop to give more process time to the processes running in the windows. The default is zero.

Startup File Format
Upon invocation MGR reads commands from the "startup file", `$HOME-/.mgrc` (see -s flag above) to initialize the display. Commands are placed one per line with the command arguments separated by spaces or tabs. The following commands are supported:

initcmd command [args...]
This command line is handed to the shell and executed at the time the startup file is read.

suspendcmd command [args...]
This command line is handed to the shell and executed each time MGR suspends it self, either due to a main menu selection or the Left-z key.

resumecmd command [args...]
This command line is handed to the shell and executed each time MGR resumes after a suspension.

quitcmd command [args...]
This command line is handed to the shell and executed just before MGR quits, either due to a main menu selection or the Left-Q key.

map n0 n1 n2 n3 n4 n5 n6 n7

This changes the meaning of the mouse buttons. Each n[0-7] represents one of the 8 states of the three mouse buttons. The default mapping is: 0 1 2 3 4 5 6 7. To change the meaning of the left and right buttons, 0 2 4 6 1 3 5 7 would be used. It is possible to map a button out of existence, which may have grave consequences.

font font_number font_name

The default font may be overridden by specifying the font name which is to be substituted for the font at position font_number. Font numbers are small integers, in the range of 0-99. The font_names are found in the font directory, by default `/usr/mgr/font`. See the -f flag above.

window x y wide high [font_number]

A window is created whose corner is at the coordinates (x, y) and whose size is (wide, high). Units are in pixels, with x and y increasing to the right and down respectively. Wide and high can be set in terms of characters in the current font by appending the letter "c" to the value. If x and y are -1, then they are replaced by values that causes new windows to "tile" across the screen. Setting wide and high to -1 is identical to setting them to "80c" and "24c" respectively. The scope of the window command continues until either another window command or done is reached. The rest of the options, shell, start, init, flags, and newwindow apply only to the current window command.

shell command [args...]

Command is the name of the command or shell to be started in the window. If command is not specified, then the environment variable $SHELL, or `/bin/csh` is used (this option applies under Sun version, not Applix).

start command

The command is sent to the shell upon startup, as if it had been typed at the keyboard.

init initial_string

The initial string is sent to the window upon startup. The string is terminated by white space, the remainder of the line may be used as a comment. The codes: \\, \b, \f \e, \n, \r, or \s may be used to represent \, backspace formfeed, escape, newline, return, or space respectively.

flags flag...

Normally a window self destructs when the original process running in it dies. if nokill, currently the only flag, is specified, the window hangs around until specifically snuffed by the user.

newwindow

The current window specification is not to be used to initialize the display, but instead will be used when Left n or Right n to create a new window.

done done must be the last line in the startup file if any window commands are specified, or the last window command will not take affect.

Using The Mouse

User interaction with MGR is with the mouse. Moving the mouse causes a corresponding movement of the mouse cursor, usually an arrow pointing to the upper left. The left or command button of mouse activates a menu whose options depend upon the current mouse position. An option is chosen by moving the mouse vertically while the command button is depressed, releasing the button when the appropriate selection is highlighted.

When the mouse is over the background pattern, or at the extreme left edge of the screen, the command menu is activated by the mouse. The command menu options are:

new window
: A new window is created by moving the the mouse cursor (now a box) to the upper left corner of the window, depressing the command button, sweeping out the window, then releasing the command button. The new window, if it is big enough, is started with a shell running in it.

redraw
: The background and windows are redrawn. This is useful if a process unknown to MGR scribbles on the display. It is left to the processing running in a window to fix the contents of its window.

quit
: MGR is terminated, after the quit is confirmed. Alternately, MGR may be suspended (ala ^Z in csh).

When the mouse is over the active window, the fat bordered window the keyboard is connected to, the window menu is activated by depressing the command button. The window menu options are:

reshape
: Reshape reshapes the active window, using a procedure similar to new window above.

move
: An outline of the current window is moved along with the mouse until the command button is depressed and released. The current window is then moved to the new location.

bury
: The current window is made inactive. Another window (if any) becomes the active window.

cut
: The mouse may be used to sweep out and save text from the current window into a global buffer. A small scissors appears as the mouse cursor. Position the upper left corner of the scissors with the upper left corner of the first character to be saved, then push one of the mouse buttons, moving the mouse to sweep out the desired text. Releasing the button causes the outlined text to be saved. Using the command button with cut causes the

current contents of the global buffer (if any) to be replaced by the indicated text. Either of the other two buttons causes the indicated text to be appended to the global buffer.

The cut facility currently works only for windows containing a single font, aligned on the default character boundaries. Applications which use only the terminal emulator sub-set of MGR capabilities, such as the shell, mail, and editors automatically meet this restriction. Cuttability may be restored by issuing a clear (i.e. form feed) to the window. The window flashes and beeps if the cut operation could not be completed, usually the result of corrupted data in the window. In such cases, no text is saved. See MGR - C Language Application Interface for a detailed description of the various cut option settings.

paste
: The contents of the global buffer (if any) are inserted into the input stream of the current window. The global buffer is filled using cut above, or under program control.

destroy
: All processes associated with the current window are sent a hangup signal, and the window is destroyed.

When the mouse is clicked on any window except the active window, that window moves to the front and becomes the active window.

Using The Left and Right Keys

When MGR is invoked from the console keyboard, many of the system menu functions have keyboard equivalents. Some of the more interesting ones are activated by holding down the Left or Right keys, and then pressing:

space bar
: to activate the previous window

Back Space
: to activate the bottom window

c
: to initiate a cut-text operation

p
: to initiate a paste operation

h
: hide the top window on the bottom

l
: to clear the active window

m
: initiate a cut-text operation which will automatically cause a paste operation when completed

n
: to start a new window, 80 x 24 characters (if it will fit), placed in the "tile" position of its window-set ID

N
: start a new window by sweeping with the mouse

Q to exit MGR quickly

1-9 to activate the window with window-set ID 1 through 9

0 activates the window with window-set ID 10, a synonym for w10<Return>

wnumber<Return>
activate the window with window-set ID number

r to redraw the windows

R to redraw the windows

The environment variable DEFAULT_FONT may be assigned the full path name of a MGR font, which will then replace MGR's built in default font.

Files

/dev/mouse place to obtain mouse coordinates.

/dev/bwtwo0 name of the display.

/usr/mgr/icon place to find MGR icons.

/usr/mgr/font place to find MGR fonts.

/usr/mgr/font/.mgrc the global default startup file; delivered with 15 fonts specified.

$HOME/.mgrc place to find startup commands.

/dev/bell For ringing the bell.

/dev/[pt]ty[pq]? Name of the pseudo-tty's.

See also

MGR - C Language Application Interface
bounce(1L) browse(1L) bury(1L) clock(1L) clock2(1L) close(1L) dmgr(1L) ether(1L) font(1L) iconmail(1L) iconmsgs(1L) loadfont(1L) maze(1L) menu(1L) mgr(1L) mgrmail(1L) mgrmsgs(1L) oclose(1L) omgrmail(1L) rotate(1L) set_console(1L) set_termcap(1L) shape(1L) show(1L) showfont(1L) snap(1L) startup(1L) stat(1L) stringart(1L) tjfilter(1L) window_print(1L) zoom(1L) bitmap(5L) font(5L)

Diagnostics

Can't find a frame buffer
No display device available. Make sure /dev/bwtwo0 exists in /dev.

Can't find a mouse, or it is already in use MGR must have exclusive control of the mouse.

Internal MGR error
everything else.

Bugs

* A separate application program, set_console(1L) is required to prevent others from scribbling on /dev/console and messing up the display.

* As MGR requires exclusive control of the mouse, it may not be invoked from within itself.

* Only fixed-width fonts are supported.

Author

Stephen A. Uhler

6
Programs Available

A considerable number of additional programs are available for use in conjunction with mgr. Some are for programmers only, while others are simple demonstrations, for impressing viewers. Details of these programs are listed hereunder, in alphabetical order. Have fun!

Bitmap - Bitmap header format for mgr bitmaps.

Synopsis

#include "dump.h"

Description

There are two styles of bitmaps recognized by MGR , The old machine dependent format, and the new portable format.

Old bitmap files are prepended with a 6 byte ascii header which contains:

1) a two byte magic number,

2) a 2 byte bitmap width, and

3) a two byte bitmap height.

The bitmap data follows the header in raster scan order, with each row padded out to a 16 bit boundary.

The new, portable bitmap format consists of an 8 byte ascii header containing:

1) a two byte magic number,

2) a 2 byte bitmap width,

3) a two byte bitmap height, a single byte bitmap depth, and 1 reserved byte.

The bitmap data follows the header in raster scan order, with each row padded to a byte boundary.

The following macros, defined in dump.h may be useful for dealing with bitmap headers:

B_HSIZE
: The bitmap header size in bytes.

B_GETHDR(header,width,height)
: extracts the width and height from the B_HSIZE buffer header

B_PUTHDR(header,width,height)
: produces a bitmap header for a bitmap width bits wide and height bits high.

B_ISHDR(header)
: returns true if header is a valid bitmap header

B_SIZE(width,height)
: returns the size in bytes (not including the header) of a bitmap width bits wide and height bits high.

B_MAGIC
: is a pointer to a character string whose first 2 bytes are the bitmap header magic number.

Bugs

The existence of two different bitmap formats is unfortunate. The old format should go away when the programs that use it are rewritten.

See also

mgr(1L)

bounce - A standard graphics demo

Synopsis

```
bounce [ -s ]
```

Description

Bounce bounces 10 lines around the window forever. The -s flag bounces the lines slower. Bounce stops if its window is obscured.

See also

mgr(1L)

Author

D. Nachbar

---

browse - An icon browser for MGR

Synopsis

```
browse filename...
```

Description

Browse displays the icon files specified on the command line in the current window. If all of the icons specified won't fit in the window, the pop-up menu accessed from the middle mouse button permits paging back and forth among the icons.

The Right or pointing button of the mouse, when clicked over an icon, highlights the icon and prints its filename.

Bugs

* The icon files are read from the same host MGR is executing on, not the host browse is running on.

See also

mgr(1L) bitmap(5L) zoom(1L)

Author

S. A. Uhler

bury - Bury a mgr window.

Synopsis

bury

Description

Bury pushes the window to the bottom of the screen.

See also

mgr(1L)

Author

S. A. Uhler

---

c_menu - Turn C error messages into vi menus.

Synopsis

```
make -k | c_menu; vi
```

Description

C_menu reads its standard input and looks for lines of the form:

```
"foo.c", line 19: word undefined.
```

All such lines are gathered into a set of menus, one for each C file, that are useful for locating the errors in the source files using the vi edittor.

The main menu contains the names of the C files with errors, selecting a file causes that file to be editted. Sliding off to the right of the file name menu pops up a list of error messages. Selecting an error message while in vi causes vi to move its cursor to the line containing the error.

The most common way to use c_menu is with the MGR cut and paste facility. After running a make or cc that produces C error messages, simply cut the error messages, type c_menu;vi , paste the errors into c_menu , then key CNTL d.

See also

mgr(1L) menu(1L)

Bugs

After adding or deleting lines from the file, c_menu's notion of which line contains the error is incorrect.

Author

S. A. Uhler

clock - Digital display of time of day on a mgr terminal.

Synopsis

```
clock [ -b ] [ -f<font> ] [ -s ]
```

Description

Clock displays a time of day clock on a mgr window. The window shrinks to just enclose the display. The -f flag is used to specify the font to use for the display. clock -f5 is typical. -b pushes the clock to the bottom of the display, and -s doesn't reshape the window.

See also

clock2(1L) mgr(1L)

Author

S. A. Uhler

---

clock2 - Analog display of time of day on a mgr terminal.

Synopsis

```
clock2
```

Description

Clock2 draws an analog clock face that fills the current window, on which it shows the current time of day. Square-ish windows produce the best results.

See also

clock(1L) mgr(1L)

Author

S. A. Uhler

close - Close a mgr window.

Synopsis

```
close [ message [ font-number ] ]
```

Description

Close Makes the current window very small, displays message in it, and moves it to an unoccupied spot on the screen. If the message includes "%d then the window set ID will replace it when the message is displayed. Upon reactivation, the window returns to its former size and position. If the closed window is ever covered, it attempts to find and move itself to some other unoccupied spot.

If no message is given or the message is of zero length, the current hostname along with the parenthesized window ID is displayed. The window ID can be used to activate the window from the keyboard. Left-windowID or Right-windowID will activate the window if the window ID is a single digit. Left-w-windowID or Right-w-windowID will activate the window for any window ID. Window ID 0 is an alias for window ID 10.

An optional second argument, fontnumber, may be specified to indicate the font in which message is displayed.

Examples

close
close ’source directory’
close ’’ 0
close ’source directory’ 7

Bugs

* Not all windows on the screen may be closed at once.

See also

mgr(1L)

Authors

S. A. Uhler
M. H. Bianchi

color - set the forground and background color for text in an Mgr window.

Synopsis

```
color [dark|light] color on [dark|light] color
```

Description

color sets the current foreground and background text color for an Mgr window. Color is one of black, white, red, green, blue, yellow, cyan or magenta. Alternately, color may be specified as an index in the color lookup table. Color calls set_colormap(1L) to initialize the color map.

See also

mgr(1L) set_colormap(1L)

Bugs

Color only works with color Mgr .

Author

S. A. Uhler

---

cut - cut text from a MGR window and send it to a program.

Synopsis

```
cut [ -s ] [ command ]
```

Description

Cut watches the global MGR cut buffer and when activated, reads the buffer and starts <command> with the contents of the buffer as <command>'s standard input.

Initially, cut prompts the user for a spot on the display, moves there and becomes a closed file cabinet icon. Any time text is cut to to MGR's global buffer, cut highlights the file cabinet to indicate it has seen the cut text. Cut prepends the current date, time and message size to the text sent to <command>.

If -s is specified, cut does not prompt the user for a spot, but uses the window as is. This is useful for starting cut from the MGR startup file. If no command is given, cut looks for the command in the environment variable CUT .

See also

mgr(1L) snap(1L)

Author

S. A. Uhler

cycle - Display a sequence of icons on an mgr terminal.

Synopsis

```
cycle [ -sspeed ] [ -r ] icon1 icon2 [ ...  <iconn> ]
```

Description

cycle will display the list of specified icons in sequence in an mgr window.

The flag -s speed sets the delay between frames in micro-seconds

The flag -r causes the frames to be run forward and then in reverse and then repeat, rather than just running them forward repeatedly.

See also

mgr(1L)

Author

S. D. Hawley

---

dmgr - A rudimentary troff previewer for mgr

Synopsis

```
ditroff [ <ditroff args> ] ...  | Dmgr
```

Description

Dmgr is a simple troff previewer for MGR . It reads ditroff output and places characters on an MGR window in their proper relative location on the page, using whatever MGR character font happens to be current. Bold face is indicated by overstiking, italics by underlining. Dmgr pauses at the end of every page and rings the bell. A RETURN causes dmgr to continue with the next page.

Bugs

* Dmgr uses the current Mgr font for output, which is probably never the appropriate font to use. As such Dmgr is useful for previewing the page layout; not for actually reading the document.

* Dmgr doesn’t know about special characters or ligatures, which are printed as dashes "-".

* Dmgr invents a page size, suitable for 8-1/2 by 11 inch printers instead of extracting it from ditroff output.

See also

mgr(1L) ditroff(1)

Author

S. A. Uhler

ether - Display a strip chart of network traffic.

Synopsis

```
ether [ -f<freq> ] [ -m<max> ]
```

Description

Ether is a graphical version of netstat that runs on mgr terminals. Ether, displays the number of input packet, output packets, and collisions on the first network interface reported by netstat.

The following options are recognized on the command line:

-f<freq>
The display is updated every freq seconds, instead of the default 3 seconds.

-m<max>
specifies the maximum number of packets counted per update. The default is 15.

See also

mgr(1L) netstat(1)

Diagnostics

Window is not wide enough Make the window wider and the graph will continue.

Window is not high enough Make the window taller and the graph will continue.

Bugs

If the window is reshaped, ether requires up to freq (usually 3) seconds to learn about the new window size.

Ether calls netstat(1) and assumes a particular output (i.e. 4.2 BSD).

Author

S. A. Uhler

font - font file format for mgr bitmaps.

Synopsis

#include "font.h"

Description

MGR font files consist of a font header followed by the bitmap data for each character. The header format is:

```
struct font_header {
unsigned char type; /* font type (magic number) '^v' */
unsigned char wide; /* character width (pixels) */
unsigned char high; /* char height (pixels) */
unsigned char baseline; /* pixels from bottom of glyph */
unsigned char count; /* number of chars in font */
char start; /* starting char in font */
};
```

All of the characters in the font are are stored in a single bitmap, wide x count pixels wide, and high pixels high.

Bugs

Only fixes width fonts are currently supported.

See also

mgr(1L) bitmap(5L)

iconmail - Notification of mail arrival

Synopsis

```
iconmail [ -s ] [ -x<pos> ] [ -y<pos> ] [ -f<font> ] [ -p<poll> ] [ -M<mailer> ]
```

Description

Iconmail looks for, and announces the arrival of new mail. When initially invoked, iconmail shrinks its window into a mailbox icon. When new mail arrives, iconmail rings the bell, displays a mailbox with mail in it. If the iconmail window is activated, usually by clicking on it with the
mouse, It either creates a larger window with readmail running in it - if you have mail, or indicates you have no mail.

While in the readmail window, the middle mouse button activates a menu of commonly used mail commands.

iconmail recognizes the following command line flags:

-s Don’t reshape the window upon iconmail invocation.

-x<pos>
Starting x-coordinate of readmail window.

-y<pos>
Starting y-coordinate of readmail window.

-f<font_number>
Font to use for readmail window.

-p<poll interval>
Look for new mail every poll interval seconds (defaults to 60).

-M<mailer>
Use mailer to read mail, instead of mail.

Bugs

The readmail window sleeps for a second at its termination to permit mail to indicate new mail arrival while reading mail.

Destroying the mail subwindow is a bad thing to do.

Files

/usr/spool/mail/$USER

See also

mail(1) mgr(1L) mgrmail(1L)

Author

S. A. Uhler

iconmsgs - message arrival notification

Synopsis

```
iconmsgs [ -s ] [ -x<pos> ] [ -y<pos> ] [ -f<font> ] [ -p<poll> ]
```

Description

iconmsgs looks for, and announces the arrival of new msgs. When initially invoked, iconmsgs shrinks its window to a bulletin board icon, displaying the number of pending messages as notes tacked to the board. When new messages arrive, iconmsgs rings the bell, and tacks a new message to the bulletin board. If the iconmsgs window is activated when messages are pending, usually by clicking on it with the mouse, iconmsgs creates a larger window with msgs running in it.

While in the msgs window, the middle mouse button activates a menu of commonly used msgs commands.

iconmsgs recognizes the following command line flags:

-s Don’t reshape the window upon iconmsgs invocation.

-x<pos>
: Starting x-coordinate of msgs window.

-y<pos>
: Starting y-coordinate of msgs window.

-f<font_number>
: Font to use for msgs window.

-p<poll interval>
: Look for new messages every poll interval seconds (defaults to 60).

Bugs

Destroying the msgs window while msgs is running is a bad thing to do.

Files

/usr/spool/msgs/bounds

$HOME/.msgsrc

See also

msgs(1) mgr(1L) mgrmsgs(1L)

Author

S. A. Uhler

invert_colormap - inverts the colormap on a SUN color display MGR .

Synopsis

```
invert_colormap
```

Description

Set_colormap inverts the current color map in the sense of a photographic negative.

See also

set_colormap(1L) overlay(1L)

Author

S. A. Uhler

---

lock - lock the sun console

Synopsis

```
lock
```

Description

lock displays a screen-phosphor saving pattern on the sun console. When you login password is typed on standard input, the screen is restored. If you mistype your password, the pattern direction reverses, and you may try again.

Files

/dev/bwtwo0 to find the screen

/etc/passwd to check the login password

See also

lockscreen(1)

Bugs

Error checking is poor

Author

S. A. Uhler

maze - A graphical game of solitare

Synopsis

```
maze
```

Description

Maze draws a maze and permits you to navigate around it while displaying both a top and perspective view. The f (or space), r, l, and b keys move you forward, right, left, and backwards respectively. You can sometimes see others in the maze if they are playing maze somewhere else on the network.

Bugs

* This is truly a mindless endeavor.

* When other maze players die, they leave ghosts in the maze.

See also

mgr(1L)

Acknowledgments

This program was written by J. Gosling for Andrew and ported to mgr.

menu - create or select an mgr pop-up menu

Synopsis

```
menu [ -options ] [ menufile(s) ]
```

Description

Menu downloads or selects a pop-up menu for the mgr window manager for a Sun workstations.

Options

-sn selects the menu previously loaded into position n; that is, it binds the menu in position n to the middle mouse button. No downloading takes place.

-s by itself allows downloading of a new menu, without it being selected.

-ln causes loading of a new menu into position n. By default, n is 1. The loaded menu is also selected (unless -s was specified).

-n "Name of Menu" specifies a name for the menu. Alternatively, a name may be specified by one or more lines of the form name=anything in the menu file. By default, the name of the menu is taken to be the name of the input file converted to upper-case. The menu's name is displayed at the top of the menu.

-n by itself suppresses the name altogether.

-fn specifies that the font loaded into font postion n will used for the menu. Default is 6.

-d<char> indicates that <char> will be used instead of the default, TAB, to delimit items from actions in the menu file.

The named input files (or standard input by default) should contain one item-action pair per line, delimited by by one or more delimiter characters (TABs by default). Items and actions are arbitrary character strings not containing the delimiter character. The items are displayed in the pop-up menu, and the user is allowed to select an item using the mouse and middle button. The corresponding action string is then written to the standard input of the process running in the window. If an action ends with \c, the newline character following the action is suppressed.

Optionally, the menu file may begin with a delim=<char> line to specify an alternate delimiter, followed by any number of name=anything lines to specify one or more lines of menu name, and a font=n line to select a font. The total number of characters in all actions and items may not exceed approximately 1000.

Examples

```
menu <<!
delim=:
name= FRUITS
name=--------------
apples:echo apples
```

```
oranges:echo oranges
pears:echo pears
passion fruit:echo passion fruit
three cherries:echo Jackpot!
!
```

Author

Paul A. Tukey (bellcore!paul)

mgrmail - Notification of mail arrival

Synopsis

```
mgrmail [ -s ] [ -x<pos> ] [ -y<pos> ] [ -f<font> ] [ -p<poll> ] [ -M<mailer>
]
```

Description

Mgrmail looks for, and announces the arrival of new mail. When initially invoked, mgrmail shrinks its window to the single line Looking for new mail. When new mail arrives, mgrmail rings the bell, and states You have new mail. If the mgrmail window is activated, usually by clicking on it with the mouse, It creates a larger window with readmail running in it.

While in the readmail window, the middle mouse button activates a menu of commonly used mail commands.

Mgrmail recognizes the following command line flags:

-s Don’t reshape the window upon mgrmail invocation.

-x<pos>
Starting x-coordinate of readmail window.

-y<pos>
Starting y-coordinate of readmail window.

-f<font_number>
Font to use for readmail window.

-p<poll interval> Look for new mail every poll interval seconds (defaults to 60).

-M<mailer>
Use mailer to read mail, instead of mail.

Bugs

The readmail window sleeps for a second at its termination to permit mail to indicate new mail arrival while reading mail.

Destroying the mail subwindow is a bad thing to do.

Files

/usr/spool/mail/$USER

See also

mail(1) mgr(1L)

Author

S. A. Uhler

mgrmsgs - message arrival notification

Synopsis

```
mgrmsgs [ -s ] [ -x<pos> ] [ -y<pos> ] [ -f<font> ] [ -p<poll> ]
```

Description

Mgrmsgs looks for, and announces the arrival of new msgs. When initially invoked, mgrmsgs shrinks its window to the single line displaying the number of pending messages. When new messages arrive, mgrmsgs rings the bell, and updates the current message count. If the mgrmsgs window is activated when messages are pending, usually by clicking on it with the mouse, It changes to a larger window with msgs running in it.

While in the msgs window, the middle mouse button activates a menu of commonly used msgs commands.

Mgrmsgs recognizes the following command line flags:

-s Don’t reshape the window upon mgrmsgs invocation.

-x<pos>
Starting x-coordinate of msgs window.

-y<pos>
Starting y-coordinate of msgs window.

-f<font_number>
Font to use for msgs window.

p<poll interval>
Look for new messages every poll interval seconds (defaults to 60).

Files

/usr/spool/msgs/bounds

$HOME/.msgsrc

See also

msgs(1) mgr(1L)

Author

S. A. Uhler

oclose - Close a mgr window.

Synopsis

```
oclose [ <message> ] [ -Fn ]
```

Description

Oclose makes the current window very small, displays message in it, and moves it to the bottom of the screen. Upon reactivating the window, it returns to its former size and position. If no message is given, the current hostname is displayed instead. An optional font number may be specified to indicate the font in which message is displayed.

Bugs

* oclose does a poor job of placing the icon.

* Not all windows on the screen may be closed at once.

See also

mgr(1L)

Author

S. A. Uhler

omgrmail - Notification of mail arrival

Synopsis

```
omgrmail [ -s ] [ -x<pos> ] [ -y<pos> ] [ -f<font> ] [ -p<poll> ] [ -M<mailer>
]
```

Description

Omgrmail looks for, and announces the arrival of new mail. When initially invoked, omgrmail shrinks its window to the single line Looking for new mail. When new mail arrives, omgrmail rings the bell, and states You have new mail. If the omgrmail window is activated, usually by clicking on it with the mouse, It changes to a larger window with readmail running in it.

While in the readmail window, the middle mouse button activates a menu of commonly used mail commands. The omgrmail window may be relocated on the screen by activating the readmail subwindow, and moving its upper left corner to the desired omgrmail window location.

Omgrmail recognizes the following command line flags:

-s Don’t reshape the window upon omgrmail invocation.

-x<pos>
Starting x-coordinate of readmail window.

-y<pos>
Starting y-coordinate of readmail window.

-f<font_number>
Font to use for readmail window.

-p<poll interval>
Look for new mail every poll interval seconds (defaults to 60).

-M<mailer>
Use mailer to read mail, instead of mail.

Bugs

The readmail window sleeps for 2 seconds at its termination to permit mail to indicate new mail arrival while reading mail.

Files

/usr/spool/mail/$USER

See also

mail(1) mgr(1L)

Author

S. A. Uhler

overlay - Enable or disable the overlay plane on a Sun 110. MGR .

Synopsis

```
overlay on|off
```

Description

Overlay enables or disables the overlay plane on Sun's that have them. Setting the overlay plane on causes the monochrome plane to obscure the color planes. Setting the overlay plane off turns off the monochrome frame buffer, permitting the color frame buffer to be visible.

See also

mgr(1L)

Author

S. A. Uhler

---

rotate - Rotate a bitmap 90 degrees.

Synopsis

```
Rotate [ -w wide -h high ] [ -x ] [ -v ]
```

Description

Rotate is a filter that rotates a 1 bit deep bitmap clockwise by 90 degrees. Normally rotate expects a bitmap in mgr format which uses a bitmap header to specify the width and height. Alternately, the -w and -h flags may be used to indicate the bitmap width and height if no bitmap header is present. If -x is specified, no bitmap header will be produced for the resultant bitmap. The -v flags prints <'s and >'s to help stave off boredom while rotate is running.

Bugs

Rotate can't currently rotate bitmaps with more than 1200 rows.

See also

bitmap(5L)

Author

S. A. Uhler

set_colormap - initialize colormap entries suitable for MGR.

Synopsis

```
set_colormap
```

Description

Set_colormap initializes the first and last 24 colormap entries suitably for color MGR . The first 8 colors are set to white, black, red, green, blue, yellow, cyan, and magenta. The second 8 colors are dark versions of the first 8 colors, whereas the third 8 colors are bright versions of the first 8 colors. The last 24 colors are set to the inverse of the first 24 colors.

See also

mgr(1L) overlay(1L)

Bugs

Set_colormap is Sun specific, and no provision is made to set the remainder of the colormap.

Author

S. A. Uhler

set_console - redirect console messages to a MGR window.

Synopsis

```
set_console
```

Description

Set_console , when run in a MGR window, redirects console messages to that window.

See also

mgr(1L)

Bugs

* Redirecting console messages raises havoc if the keyboard is not in direct mode. Set_console prints a warning message and fails if the keyboard is not in direct mode.

* Console messages automatically get reset to the console when MGR is suspended. Set_console should be reissued after resuming a suspended MGR .

Author

S. A. Uhler

set_termcap, set_emacs - set an appropriate TERMCAP entry for MGR.

Synopsis

```
eval ‘set_termcap [ -b ]‘
```

Description

Set_termcap Prints on stdout the shell commands required to set the TERMCAP environment variable appropriately for the current window size on a mgr terminal. Set_termcap looks at the SHELL environment variable to decide what shell commands are appropriate. the command eval ‘set_termcap‘ sets the TERM and TERMCAP environment variables appropriately.

Set_emacs optimizes the termcap entry for GNU-emacs, which knows how to deal with scrolling regions and multiple line inserts and deletes, whereas set_termcap keeps vi happy.

Bugs

csh usrs need to use: set noglob; eval ‘set_termcap‘ to keep the shell from getting confused.

See also

csh(1) mgr(1L) sh(1)

Author

S. A. Uhler

---

shape - Reshape mgr window.

Synopsis

```
shape [ <columns> ] [ <rows> ]
```

Description

Shape Reshapes the window to the specified number of columns and rows. With no arguments shape makes an 80 column by 24 row window. With one argument, shape changes the number of lines to the number given, leaving the number of columns unchanged.

Bugs

Given unreasonable arguments, shape doesn’t guarantee reasonable results

See also

mgr(1L)

Author

S. A. Uhler

show - displays a bit-mapped image on a mgr window.

Synopsis

```
show [-r] [ <x coord> ] [ <y coord> ] [<bits wide>]
```

Description

Show copies its standard input, which is assumed to be in MGR bitmap format to the window as a bit mapped image. The starting position of the bit map relative to the top left corner of the window may be given as x coord and y coord respectively. If bits wide is specified on the command line, show assumes no bitmap header is present. Specifying -r changes the sense of black and white.

Bit maps too big to fit on the window are clipped. The incoming data for each row should be rounded up to an even number of bytes. The bits are displayed left to right, then top to bottom.

Bugs

Large bit maps take too long to display.

See also

mgr(1L) bitmap(5L)

snap - capture a portion of the display as a bitmap image

Synopsis

```
snap [ -n ] <file>
```

Description

Snap lets a user capture a bit image of an arbitrary rectangular portion of the display. This image may be saved in a file, send to a printer, or copied back to the display.

When snap is active, the user may sweep out an image with the 3rd mouse button. Upon releasing the button, this image is captured and remembered by Snap. The middle button pops-up a menu with the following options:

Print
The image snapped is sent to the printer (via lpr) in MGR bitmap format with the -v flag set. If the PRINTER environment variable is set, it is used to specify the printer name, otherwise the image is sent to lp .

File
The last image snapped is saved in the file specified on the command line when snap was invoked, in MGR bitmap format. Successive invocations of file overwrite any previously stored images.

Review
Once Review is selected, The user may sweep out (using the 3rd mouse button, as before) an area on which to display the snapped image. If nothing is swept within 10 seconds, snap flashes, rings the bell, and reverts to capture mode. While reviewing in enabled, the snap icon remains inverse video. The displayed image is clipped to fit within the region swept out by the user. If the user simply clicks the 3rd mouse button twice without sweeping, snap copies the entire saved image to the display, starting at the mouse location.

Quit
Snap de-iconifies itself and quits.

When -n is specified on the command line, the new (8 byte header) style bitmap format is produced, instead of the old (6 byte header) format.

Files

/dev/bwtwo0 to find the display image.

See also

lpr(1) window_print(1L)

Diagnostics

* Can’t find screen The frame buffer won’t open.

* Can’t open file The file can’t be opened for writing.

Bugs

* The user interface is overly simplistic.

* snap only works on the mgr host.

* The review function can write on the display arbitrarily, destroying its integrity.

Author

S. A. Uhler

startup - produce a startup file reflecting the current mgr screen layout.

Synopsis

```
startup
```

Description

Startup produces the current mgr window layout on its standard output, in a form suitable for the mgr startup file, $HOME/.mgrc.

Bugs

startup produces only the windows positions, neglecting the fonts and commands currently running in the windows.

See also

mgr(1L)

---

stat - Display a strip chart of one or more current machine statistics.

Synopsis

```
stat [ -bsf<freq> [[ -<max> ] <parameter> ]] ...
```

Description

Stat is a graphical version of vmstat that runs on mgr terminals. Stat, with no options, displays the list of parameters it will chart.

The following options are recognized on the command line:

-b Do not update the display if the window is obscured. When the window is uncovered, the display rushes to catch up, instead of reflecting reality immediately.

-s Traces are drawn as thin lines, instead of solid filled.

-f<freq>
The time interval between display updates is freq seconds. The default is 5 seconds.

-<max>
specifies the maximum value of the following parameter, in units appropriate for that parameter.

<parameter>
is a code that represents a particular statistic to plot. The available parameters are:
r jobs in run q
b jobs blocked
w jobs waiting
fre free memory
fr freed pages

d1 disk 1 accesses
d2 disk 2 accesses
d3 disk 3 accesses
d4 disk 4 accesses
in interrupts
sy system calls
cs context switches
us % user time
kn % system time
id % idle time

See also

mgr(1L) vmstat(1)

Diagnostics

Window is not wide enough
Make the window wider and the graph will continue.

Window is not high enough
Make the window taller and the graph will continue.

Bugs

If the window is reshaped, stat requires up to freq (usually 3) seconds to learn about the new window size.

Stat calls vmstat(1), and assumes a particular (i.e. BSD 4.2) output format from vmstat(1).

stringart - A standard graphics demo

Synopsis

```
stringart
```

Description

Stringart draws bunches of lines in a geometric patterns, erases them, and starts again with a different pattern.

See also

mgr(1L)

---

tjfilter - Bitmap lpr filter for the HP ThinkJet printer.

Synopsis

```
tjfilter [ -r ]
```

Description

Tjfilter reads its standard input which is expected to be a one bit deep bitmap image in mgr format and transforms it for printing on an HP ThinkJet printer. If -r is specified, the bitmap is reversed (ala a photographic negative). For bitmaps which are wider than 640 dots, but less than 640 dots high, tjfilter calls rotate(1L) in order to fit the bitmap on the printer.

Tjfilter is normally used as the vf filter in the HP ThinkJet printcap entry:
...:vf=/usr/local/bin/tjfilter:...

Bugs

The HP Thinkjet printer can print a maximum of 640 dots per line. Bitmaps that don’t fit get truncated.

See also

bitmap(5L) rotate(1L) lpr(1L)

window_print - print an image of an MGR window on a printer.

Synopsis

```
window_print [ -f filter ] [ -j name ] [ -m message ] [ -p printer ] [ -o
option ] [ -x file ]
```

Description

Window print prints the images of windows on a printer. When first invoked, Window print iconifies itself with the message Window dump. A hard copy of a window is made by activating the window print window and clicking the third mouse button over the desired window. Window_print copies the image of the window onto a file, then invokes lpr to print it.

command options

-f filter
: The name of an (optional) unix filter for converting the mgr bitmap format file into a form suitable for lpr.

-[jJ] name
: The name printed on the burst page of the printer, normally window.

-m message
: The string displayed in the iconified window.

-[pP] printer
: The name of the printer, as in lpr.

-v option
: Normally lpr is invoked with the -v flag. If option is specified, it is used instead.

-x file
: The image files is copied to file instead of being sent to lpr

Files

/tmp/pr* temporary bit image file

See also

lpr(1)

Diagnostics

debugging output may be obtained by setting the environment variable DEBUG.

Bugs

The temporary file is created on the machine which is running mgr, not the machine running window_print.

zoom - an icon editor for mgr

Synopsis

```
zoom <icon file> ...
```

Description

zoom is a mouse driven icon editter for mgr. Zoom divides the window into three regions, a banner line at the top containing four status fields, a message line at the bottom, and the remainder of the window for an enlarged, or zoomed representation of <icon> being editted. If the first file given on the command line is not an icon, zoom prompts for its width and height.

The current state of zoom is indicated by the four status fields in the banner line.

* The first, or raster function field displays the current raster-op function to be applied to the next edit operation. This function may be changed with the pop-up menu, activated by pressing the middle mouse button while the mouse track is in the raster function field. Normally the choices are set, clear, toggle and grid. The first three are raster-op functions; the grid option toggles the bitmap alignment grid. If the put command is pending (see below), the raster-op choices become copy, paint mask, and exclusive-or.

* The second, or edit field displays one of the six possible edit functions: Yank, Put, Shrink, Grow, Fix, and Undo. Fix and Undo are performed when selected. Fix changes the window size to give square pixels. Undo un-does the previous edit operation. If any of the other functions is selected, it becomes the pending function, and is highlighted. When a function is pending, the next sweep operation performs that function on the group of pixels enclosed by the sweeping rectangle (the selected pixels).

Yank copies the selected pixels into the yank buffer.

Put combines the yank buffer with the selected pixels in a manner determined by the current raster function field.

Shrink makes the icon smaller by scaling the selected pixels to fill the entire window.

Grow makes the icon bigger by scaling the entire icon to fit into the selected pixel region.

* The Third or size field displays the current width and height of the icon, in pixels. The size of the icon may be changed by selecting the pop-up menu when the mouse track is in the size field and responding to the prompt. While in the prompt window, the menu permits the selection of several standard icon sizes.

* The fourth, and final field is the file field. The file field displays the current file name of the icon. The filing options Save, Get, Yank, and Quit are, as usual,

accessed by a pop-up menu when the mouse track is in the file field. The file options prompt for a file name. A list of all of the files specified on the command line is available via the pop-up menu within the prompt window.

Save saves the icon by the specified name.

Get edits a new icon , tossing the current icon into the bit-bucket.

Yank copies the specified icon into the yank buffer for use with the put command.

Quit quits Zoom. Quit does NOT save the icon. A save must be explicitly issued first. Zoom may also be terminated by typing "Q\r" to the window, or hitting your favorite interrupt key.

For those who are not particularly font of rodents, all of the zoom commands may be accessed via 1 or 2 letter keyboard commands (followed by a \r), some of which are:

R Repaint window
x toggle alignment grid
w FIX window aspect ratio
u UNDO

s1 select SET mode
s2 select CLEAR mode
s3 select TOGGLE mode

f SAVE file
g GET a new file
y YANK a file
Q QUIT

F1 select YANK function
F2 select PUT function
F3 SHRINK icon
F4 GROW icon

P0 set COPY mode
P1 set PAINT mode
P2 set MASK mode
P3 set XOR mode

Okay, now to edit the icon.

* Pressing the middle button and moving it either sets or clears the pixels it passes over. If the first pixel it touches is clear, the pixels will be set; if it is set, all touched pixels will be cleared.

* Holding, moving, then releasing the right mouse button sweeps out a rectangular region of selected pixels. If no function is currently highlighted in the edit field, the current raster-op function is performed on the selected pixels. Otherwise, the highlighted function is performed.

Bugs

* Zoom works best on small icons, running on the local machine.

* You can’t view the actual size of the icon being editted.

* Icon coordinates must be typed in exactly in the form of x , y with no spaces or tabs.

See also

browse(1L) dump(5L) mgr(1L)

7
MGR - C Language Application Interface

Stephen A. Uhler, Bell Communications Research

Introduction

MGR (manager) is a window system for Unix that currently runs on Sun Workstations and the Applix 1616. MGR manages asynchronous updates of overlapping windows and provides application support for a heterogeneous network environment, i.e., many different types of computers connected by various communications media. The application interface enables applications (called client programs) to be written in a variety of programming languages, and run on different operating systems. The client program can take full advantage of the windowing capabilities regardless of the type of connection to the workstation running MGR.

Client programs communicate with MGR via pseudo-terminals over a reliable byte stream. Each client program can create and manipulate one or more windows on the display, with commands and data to the various windows multiplexed over the same connection. MGR provides ASCII terminal emulation and takes responsibility for maintaining the integrity of the window contents when parts of windows become obscured and subsequently uncovered. This permits naive applications to work without modification by providing a default environment that appears to be an ordinary terminal.

In addition to terminal emulation, MGR provides each client window with: graphics primitives such as line and circle drawing; facilities for manipulating bitmaps, fonts, icons, and pop-up menus; commands to reshape and position windows; and a message passing facility enabling client programs to rendezvous and exchange messages. Client programs may ask to be informed when a change in the window system occurs, such as a reshaped window, a pushed mouse button, or a message sent from another client program. These changes are called events. MGR notifies a client program of an event by sending it an ASCII character string in a format specified by the client program. Existing applications can be integrated into the windowing environment without modification by having MGR imitate keystrokes in response to user defined menus or other events.

The user interface provides a simple point-and-select model of interaction using the mouse with pop-up menus and quick access to system functions through meta-keys on the keyboard. MGR also provides a cut and paste function that permits a user to sweep out and copy text from any window and paste it into any other.

This document describes the low level C interface library for MGR. The C Interface library provides a set of macros and functions which implement the stream protocol and provide clients written in C with a function call interface to MGR. This library provides the lowest level access to MGR functions and represents a direct mapping to the underlying protocol. It is expected that a higher level interface will evolve to support application development at a higher level. The library requires only the UNIX Standard I/O Library for its operation and access to a byte sequential I/O interface from the underlying operating system.

Model of Interaction

The basic unit within MGR is the window. A window is a rectangular region on the display, surrounded by a border, with a single connection to other processes. All interactions among the client program, the user and MGR are defined entirely in terms of the state of a client's window or windows. MGR has no concept of window types; there are no separate graphics windows, text windows, or edit windows. Every window supports exactly the same set of capabilities as every other window. In addition, all windows act independently. Client programs need not know or care about the existence of other clients or windows that happen to coexist on the same display. The management of overlapping windows is handled entirely by MGR. For example, when a window is partially or totally obscured by another window, then subsequently uncovered, MGR restores the integrity of the window's contents. There are no sub-windows, windows whose size or position are in some way restricted by a parent window. A client may create and manipulate many windows, each of which may be positioned and sized independently on the display.

At any given time there is one special window on the display, the active window. This is the window that receives keystrokes and mouse data. It is distinguishable to the user from the other windows on the display by its emboldened border. The active window, in addition to receiving all mouse and keyboard data, is also logically in front of the other windows on the display. The active window is, therefore, always completely exposed. Any window can become the active window, but there can only be one active window at a time.

A client program may change its window at any time, write text into it, draw lines, anything, so long as the change is local, that is the change affects just its window. Only the active window may effect global changes to the display, such as changing its shape or position. The only global action a non-active window may perform is to become the active window. This window model provides both the user and application developer with a simple, consistent model of interaction.

Coordinate Systems

MGR uses four different coordinate systems, display coordinates, absolute window coordinates, relative window coordinates, and character coordinates. The entire display is represented by display coordinates whereas each window has its own absolute window, relative window, and character coordinate systems.

Display coordinates are in units of pixels. The coordinate (0,0)is the top left pixel on the display. The X coordinate increases to the right, the Y coordinate increases down. The maximum X and Y coordinate depend upon the particular display in use, for the SUN-3 they are 1152 by 900. Commands that operate on the context of the entire display, such as reshaping a window are specified in display coordinates. Windows, when measured in display coordinates include their borders.

Absolute window coordinates, as with display coordinates, are measured in units of pixels. The X and Y values increase to the right and down respectively. The origin, coordinate (0,0)is at the top left corner of the window, just inside the window border.

Relative window coordinates are measured as a fraction of the window's size and shape. As with absolute window coordinates, each window has its origin, (0,0), at the top left corner of the window just inside the border, however the lower right corner of the window is always at coordinate (999,999). Graphics commands to a window in relative window coordinates are automatically scaled to the size of the window.

Character coordinates are measured in rows and columns in the current font, just like an ordinary terminal. The coordinate (0,0)is the top left character position in the window. The maximum row and column in the window depends on both the window and font size.

Functional Overview

The types of commands a client program may issue MGR are divided into 14 categories: terminal emulation, graphics, bit-blts, window positioning, font changes, state inquiry, saved contexts, menus, events, sweep functions, multiple window manipulation, cut and paste, messages, and window modes. What follows is a brief description of those command categories, and some examples of specific functions within the category. A detailed description of each command is provided in the following section.

Terminal Emulation

At its basic level, every MGR window emulates a CRT terminal. It provides functions for inserting and deleting lines and characters, highlighting text, clearing areas and windows, and arbitrary cursor motion capabilities. Sample MGR

TERMCAP and TERMINFO descriptions are given in the tables below. No entries are provided for keyboard key values, as they depend upon the particular keyboard in use.

Sample MGR TERMCAP Entry _ Px MGR MGR terminal:\
:am:bs:im=:ta=^I:\
:AL=\E%da:al=\Ea:\
:cd=\EC:ce=\Ec:cl=^L:\
:cm=\E%r%d,%dM:\
:co#80:li#24:\
:cs=\E%d,%dt:\
:DC=\E%dE:dc=\EE:\
:DL=\E%dd:dl=\Ed:\
:do=\Ef:up=\Eu:nd=\Er:\
:IC=\E%dA:ic=\EA:\
:se=\En:so=\Ei:\
:ve=\Ev:vs=\EV:

Sample MGR TERMINFO Entry _ Px | MGR | MGR Terminal,
cols#80, lines#24,
am, msgr, ht=^I,
clear=^L, cr=^M, bel=^G,
cub1=^H, cud1=\Ef, cuf1=\Er,
cuu1=\Eu, ind=^J,
cup=\E%p2%d;%p1%dM,
csr=\E%p1%d;%p2%dt,
wind=\E%p2%d;%p2%p4%+%d;%p1;%p1%p3%+%d;t,
el=\Ec, ed=\EC,
il1=\Ea, dl1=\Ed,
il=\E%p1%da, dl=\E%p1%dd,
smso=\Ei, rmso=\En,
smcup=\E1664P, rmcup=\Et\Ep,

MGR permits the client program to restrict the terminal emulator to an arbitrary subrectangle within the window, called a text region. For example, a text editor wishing to provide scroll bars or banner lines can still let MGR do the terminal emulation by specifying a text region that excludes the top and sides of the window. This text region may be redefined or moved around at will, permitting multiple terminal sub regions in the same window.

Graphics

In addition to terminal emulation, MGR provides a suite of pen plotter style graphics primitives. A client program may draw lines, circles, ellipses, and elliptical arcs on a window. The graphics objects may either be completely positioned, or located relative to an internal graphics point, maintained by MGR. The objects may also be drawn into undisplayed or scratchpad areas, then copied to the window as a single unit. The graphics point may be aligned with the character

cursor, for locating graphic objects relative to character text. Conversely, the character cursor may be aligned with the graphics cursor, permitting character text to be placed at arbitrary positions on the window.

Bit-blts

MGR provides a complete set of functions for dealing with bitmaps, or rectangular arrays of pixels. Bitmaps may be combined with any of the 16 possible bit-blt operations. Non-displayed bitmaps of arbitrary size may be created and destroyed, and bit-blts may be performed on the window, within a scratch-pad bitmap, between two scratch-pad bitmaps, or between a scratch-pad bitmap and the window. Bitmap images may be down-loaded from client programs to MGR, or up-loaded from MGR to the client program. In addition, bitmaps may be saved in files by MGR, or loaded into MGR from files. These last two capabilities permit client programs to manipulate large amounts of bitmap data without the need to send the bits over the communication channel.

Window Positioning

Either the user or client program may move the active window around on the display. Windows may be moved with their size retained, reshaped but remain at the same location, or be both moved and shaped anywhere on the display. If the window is the active window, it may be buried (shoved to the back on the display). If the window is not the active window, it can become the active window and then moved about on the display.

Font Changes

Client programs may change character fonts at any time, even on a character by character basis. MGR comes with scores of different fonts, ranging in size from microscopic to viewgraph size. Client programs are free to create and down-load their own fonts. The fonts supplied by MGR are constant width, that is i's take up the same amount of room as m's do. There are commands to aid client programs that wish to use proportional fonts.

State Inquiry

A client program may ask MGR about the state of its current window, such as its size and position on the display, the name and size of the current font, the position and extent of the text region, and the state of various mode settings. The client may also inquire about the state of the window system as a whole. That includes the position and state of the mouse, the number and sizes of the available fonts, and the organization of windows on the display. The display organization may include the position, size, name, ownership, and spatial ordering for all windows on the display.

Saved Contexts

Certain parts of the current window environment may be pushed on a stack, then restored at some later time. Client programs rarely need to know the context in

which they are called. They simply push those aspects of the environment they will change, then restore them before exiting. About a dozen different parts of the window environment, such as menus, character fonts, window position, etc. may be stacked independently, or in any combination.

Menus

MGR has built in support for pop-up menus. Clients may arrange for menus to pop-up in response to mouse button hits. Up to 50 menus may be down-loaded at once for each window. The client selects which menu will pop-up when a mouse button is pushed. When an item of a pop-up menu is chosen, MGR returns the string previously put into the menu by the client program. The client program may arrange for different menus to pop up depending upon the current mouse position. Menus may also be linked together as a pop-up menu tree. Sliding off to the right of a menu (called a parent menu) while an item is selected can cause another menu (called a child menu) to pop up. Any item of the parent menu may be specified as the entry item for a child menu. Upon selecting an item of a child menu, the client program may arrange for MGR to return ether the action string associated with just the child menu item, or the action strings for the selected items of all the menus. Similar to sliding menus, MGR supports paging menus as well. Long menus may be broken into several pages by the client program. MGR manages the paging automatically, popping up the next page as the user slides off the bottom of a paged menu.

Events

Client programs may arrange to be informed by MGR when some change, called an event, happens to the state of the window system. As with menus, the message informing the client program of a change is formated as specified by the client program. Examples of events include mouse buttons being depressed or released, windows changing shape or moving, and the window becoming the active window or being covered by another window. Window state information, such as the current cursor position, may be returned as part of an event string.

Sweep Functions

It is often convenient for client programs to sweep, or rubber-band simple objects, such as lines or boxes, in response to moving the mouse. MGR provides client programs with a mouse activated sweep function. MGR tracks an edge of the line or box with the mouse and reports the coordinates to the client at the conclusion of the sweep operation, when the user releases the mouse. As usual, the client program specifies the format of the data returned by MGR.

Multiple Window Manipulation

A single client program may create and manipulate additional windows, called alternate windows. The data destined for, or to be received from, an alternate window is multiplexed on the same channel as the main window. The client program selects a window to receive output, and all output goes to the selected

window until a different window is selected. For input, the client program uses the event mechanism to determine from which window input arrived. Alternate windows have the same capabilities as the main window. There is currently no limit to the number of alternate windows a client program may have. Up to 100 windows may exist on the display at one time before performance begins to degrade seriously.

Cut and Paste

MGR provides a globally accessible snarf buffer shared among all client programs. Any client program may put data into or read data from this buffer. MGR provides a user initiated cut and paste function from the command menu.

MGR extracts character text from the window and places its ASCII representation into the snarf buffer. Paste copies the contents of the snarf buffer to the input stream of the active window. Client programs, by manipulating the data in the snarf buffer, can interact with the system cut and paste functions.

Messages

Although the snarf buffer gives client programs a simple asynchronous interprocess communication mechanism, MGR has a more general synchronous interprocess message passing scheme. A client program may send a message to another client program, or broadcast the message to all client programs. As a message recipient, the client program may elect to receive messages as an event and encapsulate the message and sender name in the format of its choice. MGR provides the primitives needed to implement server clients by permitting servers to register their names, services and protocols with MGR. Client programs may query MGR for a list of active servers. Server messages may be associated with windows by the server client programs in such a way that the message is automatically received by a client program as part of a mouse button event whenever the mouse button is pressed on the server's window. Using this mechanism, client programs can interact with server clients without any advance knowledge of which servers are available or what services they are providing.

Window Modes

Client programs may select various combinations of window modes. These modes tailor the behavior of the macros described above. Examples of window modes include auto line wrap and character overstrike that affect the terminal emulation, different coordinate system settings that affect graphics commands, or flags that set a window to activate automatically upon receiving input, ignore all keyboard input, or suspend output while a window is obscured.

Underlying Protocol

The purpose of this library package is both to provide a function call interface to the stream protocol, and to document each command understood by MGR. There are two types of MGR commands, as summarized in the table below.

MGR command protocol _ T{ ESC X1, X2,...,

Xn command T} T{ ESC X1, X2,..., Xn length command data T}

In both cases, ESC is the ASCII escape character or '\033', whereas the word command represents a single character command identifier. The X's are optional integers, there can be as few as zero, as in the command center box; c. ESCa which inserts a blank line in the window, or as many as eight, as would be used by the command

ESC0,0,50,100,10,20,3,2b

which is an example of a command to copy images between bitmaps. No spaces may be included between the ESC character and the command identifier character, but the argument separators may be either commas (,) or semicolons (;).

The function of the command is determined both by the command identifier character and n, the number of numeric arguments preceding the command identifier character. All of the commands with the same command identifier character are closely related in function. For example, all the commands in the following table have the same command character, 'o', and all draw ellipses, but have different effects based upon the number of arguments.

Commands that draw ellipses

1 ESC100,200o

2 ESC100,200,300,400o

3 ESC100,200,300,400,2o

All of the ellipses have major and minor axis lengths of 100 and 200 units respectively. Command 1 draws the ellipse at the current graphics location. Command 2 draws the ellipse at the location specified by the third and forth arguments, at (300,400). Command 3 draws the ellipse into scratchpad bitmap number 2.

The second form of MGR commands, which is a special case of the first form, is used for downloading data from the client program to MGR. The integer length specifies the number of bytes of data to be downloaded, and data are the length number of data values downloaded. An example of the second type of MGR command is

ESC11,7bI-moved

which instructs MGR to send the client program the string I-moved any time the client's window is moved to a different location on the display. The 11 refers to the number of the move event and the 7 is the number of characters in the event string, which in this case is I-moved.

All of the command identifier characters are listed in window.h. The command actions, determined by the command identifier and number of command arguments, are described by the macros in this document.

Conventions and Notation

All functions and macros and programming examples are shown in a typewriter font to distinguish them from ordinary text. Similarly, function and macro arguments are shown in a bold typewriter font.

The names of often used arguments passed to macros indicate their function, and are defined below.

column,row
: The integers column and row refer to a character position in character coordinates even though characters may be placed at arbitrary pixel locations within a window and need not fall on column or row boundaries.

Dwidth,Dheight
: The integers Dwidth and Dheight represent a width and height in display coordinates.

mode
: The positive integer mode, represents the bit combination of window modes. Mode is usually an ored list of constants in term.h. A typical use of mode is the argument to m_push(mode) as in m_push(P_FLAGS | P_EVENT | P_MENU).

n
: The small non-negative integer n represents a resource descriptor when describing objects such as windows, fonts, or menus.

name
: Name is the file name of a bitmap image on the MGR-host machine. File names given with no directory prefix are referenced relative to the icon subdirectory of MGR's home directory. The home directory is installation dependent, and may be determined with the command MGR -V.

parent,child
: The small positive integers parent and child represent menus. A child menu is linked to a parent menu forming a tree of menus.

radius
: The positive integer radius along with radius1 and radius2 signifies a radius when referring to circles or major and minor axis when referring to ellipses. They are only referenced in respect to window coordinates.

string
: An array of characters, string is a null terminated ASCII character string. Except where noted, several ASCII control characters can be included in strings by escaping them with \X, where X is one of the characters shown in the following table.

Character string control characters

escape	octal	Meaning character value =
\b	010	Back space
\E	033	Escape
\e	033	Escape
\f	014	Form feed
\g	007	Bell
\M	*	Turn on 8'th (parity) bit
\n	012	New line
\r	015	Return
\s	040	Space
\\	134	Back-slash (\)

* (the next character has its 8'th bit turned on)

to,from

The small positive integers to and from identify the destination and source bitmaps for bit-blt operations. The value 0 (zero) represents the current window bitmap; positive integers name scratch-pad bitmap storage.

width,height

The integers width and height represent a width and height in window coordinates.

X,Y

The integer pair (X,Y) represents a point in display coordinates. The suffixes src and dst as in (X_src,Y_src) or (X_dst,Y_dst) are used to indicate source and destination coordinates respectively. Similarly, the suffixes 1 and 2 as in (X1,Y1) refer generically to the first or second coordinate.

x,y

The integers (x,y) represent a point in window coordinates. Whether that is relative (i.e. 0-999) or absolute depends upon the current coordinate setting of the window. As with (X,Y) above, the modifiers src,dst,1, and 2 refer respectively to the source, destination, first, and second coordinates.

Macros

All of the C library interface macros expand into printf expressions that convert their command specification into the MGR stream protocol. Compile time and run time options are available that globally alter the behavior of these macros to the specific needs of the client program. The options are detailed in the Using the Library section of this document. The returned value of the macro expressions are not meaningful. The macros described here attempt to reflect the actual state of the system, and may include some inconsistencies that should be cleaned up in

future releases of the software. Every MGR command (a command identifier - argument count combination) that is accepted by MGR has a macro describing its function.

m_addchar()
: Inserts a space character at the current character cursor position. The remaining characters on the line, if any, are shifted to the right.

m_addchars(n)
: Inserts n space characters at the current character cursor position. The remaining characters on the line, if any, are shifted to the right.

m_addline()
: Inserts a blank line at current row. The current row, and any below it, are shifted down one line. The bottom line of text is scrolled off the window.

m_addlines(n)
: Inserts n blank lines at current row. The current row, and any below it are shifted down. The bottom n lines of text are scrolled off the window. It is much more efficient to call m_addlines(n) once, than it is to call m_addline()n times.

m_aligntext()
: Moves the current character cursor coincident with the current graphics point. The current graphics point is set with m_go (x,y). This permits client programs to position characters at arbitrary pixel locations on the window.

m_arc(x,y,x1,y1,x2,y2)
: An arc centered at (x,y) is drawn counter clockwise from (x1,y1) to (x2,y2) using the current drawing function (see m_func()).

m_bell()
: The window flashes and the bell, if there is one, rings. Even if the window is totally obscured, its flashing is made visible to the user.

m_bcolor(color)
: The background color for text operations is set to color, which is an index into the color lookup table. This command is ignored on a monochrome display. See also m_fcolor()and m_linecolor().

m_bitcopy(x_dst,y_dst,wide,high,x_src,y_src)
: Copy a rectangle from one place on the window to another with the copy function set by m_func(n). The rectangular area at x_src,y_src of size wide by high is combined with the rectangle at x_dst,y_dst according to the last function set by m_func(). The resultant rectangle is placed at x_dst,y_dst.

m_bitcopyto(x_dst,y_dst,wide,high,x_src,y_src,to,from)

Combine the rectangle at position x_src, y_src of size wide by high of bitmap from with the rectangle of the same size at x_dst ,y_dst of bitmap to. The bit-blt function used to combine the two rectangles is set by m_func(). From and to are scratch-pad bitmap descriptors. Scratchpad bitmap 0 (zero) represents the current window contents and may be used for the source , destination or both. If the scratchpad bitmap to does not already exist, it is created with a size of wide+x_src by high+y_src.

m_bitcreate(n,wide,high)

Create the scratchpad bitmap n of size wide by high. The contents of the bitmap are undefined. The macro m_bitwriteto() can be used to initialize the scratchpad bitmap. If the bitmap already exists, the old one is first discarded. Scratchpad bitmaps may also be created implicitly by specifying a non-existent bitmap as the destination of m_bitwriteto(), m_bitcopyto(), m_bitld(), or m_stringto().

m_bitdestroy(n)

Destroys scratchpad bitmap n and frees all the resources associated with it. If n does not exist, this macro is ignored.

m_bitfromfile(to,name)

Copy the file name on the MGR-host machine into the scratchpad bitmap to. The scratchpad bitmap to is created if it does not already exist. MGR returns a single line containing the width and height of the bitmap, or a blank if the file could not be found or loaded. If name does not start with / or ./, The file name is prefixed with MGR's home directory and /icon/. Names starting with ./ are evaluated relative to the directory current when MGR was invoked. The format of the bitmap file is described in dump.h.

m_bitget(from,size,offset)

Upload part of a scratchpad bitmap. Size bytes of scratchpad bitmap from, starting offset bytes from the beginning of the bitmap are sent to the client program from MGR. After this call, the client is expected to read size bytes from the input stream. The image data is sent in raster scan order, 8 pixels to a byte, padded at the right of every line to 16 bits. Size should be kept small (about 80 bytes), to avoid potential flow control problems, with bitmaps uploaded in multiple passes. (See also m_bitsave()). The data sent by MGR for this macro requires an eight bit data channel, so its use is discouraged. The macro m_bitfromfile() should be used instead when possible.

m_bitld(wide,high,x,y,size)

Prepare MGR to download a bitmap to the window. This macro instructs MGR that the next size bytes received will be interpreted as a bitmap image to be displayed on the window starting at location (x,y), and of size wide by high. Downloading bitmaps requires an eight bit channel between MGR and the client program. Large bitmaps are best sent in several pieces. The macro m_bitfromfile()should be used instead, where

possible, as it only requires a seven bit data channel and avoids the movement of large amounts of data through the channel. If more bytes are specified than required by the size of the bitmap, they are discarded. If the number specified by size is insufficient to fill the entire bitmap, the remainder of the bits are set to undetermined values.

m_bitldto(wide,high,x_dst,y_dst,to,size)

Prepare to download a bitmap to the scratchpad bitmap to. If to does not already exist, (see m_bitcreate()), it is created automatically with size wide by high. This function instructs MGR that the next size bytes received will be interpreted as a bitmap image to be copied to scratchpad bitmap to at location x_dst,y_dst. This function requires an eight bit channel between MGR and the client program. If more bytes are specified than required by the size of the bitmap, they are discarded. If the number specified by size is insufficient to fill the entire bitmap, the remainder of the bits are set to undetermined values.

m_bitsave(from,name)

Save a scratchpad bitmap from on the file name on the MGR-host file system. If name does not start with / or ./, name is prefixed with MGR's home directory and /icon/. Names starting with ./ are evaluated relative to the directory current when MGR was invoked. Specifying from to be 0 (zero) saves the entire display contents to the file. The functions m_windowsave() and m_othersave() are used to save contents of entire windows.

m_bitwrite(x_dst,y_dst,wide,high)

The rectangular region of the window, starting at x_dst,y_dst of extent wide by high, is set, cleared or inverted as determined by the previous call to m_func().

m_bitwriteto(x_dst,y_dst,wide,high,to)

The rectangular region of scratchpad bitmap to, starting at x_dst,y_dst of extent wide by high, is set, cleared or inverted as determined by the previous call to m_func(). If the destination to does not exist, it is created with the dimensions wide by high.

m_broadcast(string)

The message string is broadcast to all windows that are listening. Listening is turned on by setting the ACCEPT event for a window. (See m_setevent()). Messages can only be sent to windows whose controlling terminals have general write permission disabled (i.e. mode 0400) as a security measure to prevent malicious foreign processes from sending shells messages that get interpreted and executed as commands.

m_circle(x_dst,y_dst,radius)

A circle of radius radius is drawn, centered at x_dst,y_dst. The points at the edge of the circle are set, cleared or inverted depending upon the last call to m_func(). Circles are never scaled, they always appear as circles

on the display, regardless of the window shape. The radius is scaled based upon the average width and height of the window. Use m_ellipse()to obtain a scaled circle.

m_clear()

The current text region is cleared, and the character cursor is moved to position (0,0). If no text region is set, m_clear() clears the entire window.

m_cleareol()

All of the characters on the current line, starting with the current character to the end of the text region, are cleared. If no text region is set, m_cleareol() clears to the edge of the window.

m_clearеos()

All of the characters in the current text region, from the current character to the end of the text region, are cleared. If no text region is set, m_clear() clears to the end of the window.

m_clearevent(n)

Event n is cleared. The integer n is one of: ACCEPT, ACTIVATE, BUTTON_1, BUTTON_1U, BUTTON_2, BUTTON_2U, COVERED, DEACTIVATE, DESTROY, MOVE, NOTIFY, PASTE, REDRAW, RESHAPE, SNARFED, STACK, or UNCOVERED. See m_setevent() for a description of the events.

m_clearmenu(n)

If menu n exists, it is cleared.

m_clearmode(mode)

Clear one of the following window modes. Except where noted, these are the default settings. The mode settings are more fully explained in m_setmode().

M_ABS

Absolute window coordinate mode is turned off. The window is now in relative window coordinates. All window coordinates range from (0,0) in the upper left corner to (999,999) at the lower right.

M_ACTIVATE

Bury the window. Unlike the other window modes, bury has no state associated with it, just a one time action. A window is buried by visually pushing it to the bottom of the display; any window intersecting it will appear in front of it.

M_AUTOEXPOSE

Do not automatically activate the window the next time it receives output.

M_BACKGROUND

MGR does not permit the window to update if it is partially or totally obscured. Data destined for the window is held until the window is uncovered, or M_BACKGROUND is turned on.

M_NOINPUT
Accept keyboard input if the window is active.

M_NOWRAP
Wrap the character cursor to the next line when it reaches the right margin of the text region.

M_OVERSTRIKE
Do not overstrike characters. Text is copied to the window as if m_func(B_SRC) is set.

M_SNARFHARD
The system cut function fails if any errors are found. This is indicated by flashing the window. The contents of the cut buffer are left undisturbed. Normally this only happens if some process unknown to MGR scribbles into the region being cut.

M_SNARFLINES
Sets the system cut function to cut individual characters, instead of entire lines only.

M_SNARFTABS
The system cut function cuts text exactly as it appears on the window. All interior white space is converted to spaces, all trailing white space is treated as a new line.

M_STACK
Event stacking is turned off. Event strings are only returned for the current window context and not for any contexts stacked with m_push().

M_STANDOUT
Characters are drawn in black with a white background.

M_WOB
The window foreground color is set to black, and the window background is set color to white.

M_DUPKEY
The keyboard escape character is turned off. This mode is turned on by calling m_dupkey(). There is no corresponding call to m_setmode().

m_deletechar()
The character at the cursor position is deleted. Any characters on the line to the right of the cursor are shifted left one character position. The last character on the line is set to a space.

m_deletechars(n)
The next n characters are deleted, starting at the character cursor position. Any characters on the line to the right of the cursor are shifted left n character positions. The last n characters on the line are set to space.

m_deleteline()
The line at the cursor position is deleted. Any lines below the cursor are shifted up one line. The last line is cleared.

m_deletelines(n)

The next n lines starting at the cursor position and toward the bottom of the window are deleted. Any lines below the deleted ones are shifted up n lines. The last n lines are cleared. It is more efficient to make one call to m_deletelines() than to call m_deleteline()n times.

m_destroywin(n)

Destroy alternate window n, created by calling m_newwin(). If output is currently directed to this window, it is automatically re-directed to the main window. See also m_selectwin().

m_down(n)

Move the character cursor down n tenths of a character height. This may cause the window to scroll. See also m_left()m_right()and m_up().

m_draw(x,y)

Draw a line from current graphics point to (x,y). The macro m_go() is used to move the graphics point. The graphics point is left at (x,y), the end point of the line.

m_dupkey(char)

Every time the character char is typed at the keyboard for this window, it is sent to the client program twice. This enables clients to reliably distinguish keyboard input from that generated by menu selections or events by starting every event and menu string with a two character code whose first character is char and whose second character is anything but char.

m_ellipse(x,y,radius1,radius2)

Draw an ellipse centered at (x,y). The values for radius1 and radius2 are the major and minor axis radii. The ellipse is either set, cleared or inverted determined by the last call to m_func(). The values for radius1 and radius2 are scaled based upon the average width and height of the window.

m_ellipseto(to,x,y,radius1,radius2)

Draw an ellipse on scratchpad bitmap to centered at (x,y) where radius1,radius2 are the major and minor axis radii. The ellipse is either set, cleared or inverted determined by the last call to m_func(). The values for radius1 and radius2 are scaled based upon the average width and height of the window. If the offscreen bitmap to does not exist, this call is ignored.

m_fastdraw(x,y,count,data)

The next count bytes of data are sent to MGR are to be interpreted as lines drawn in fast draw mode, starting at (x,y). Fast draw mode permits the rapid drawing of short vectors by encoding an x,y displacement location in a single byte. The x coordinate is contained in the most significant 4 bits, the y coordinate in the least significant 4 bits. Values for x and y represent displacements from the previous location, and range

from +7 to -8. A 7 is coded as 0xff, a -8 as 0x00. If both x and y are zero (i.e. 0x8080). The next coordinate is taken to be a move instead of a draw. An eight bit channel between MGR and the client program is required for fast draw mode. See also m_rfastdraw().

m_fcolor(color)

The foreground color for text operations is set to color , which is an index into the color lookup table. This command is ignored on a monochrome display. See also m_bcolor()and m_linecolor().

m_flush()

Flush the MGR output buffer. This is equivalent to the stdio function fflush() and is needed only when the M_FLUSH flag is not specified in the call to m_setup().

m_font(n)

Change to font n. The line positioning is adjusted to keep the baseline of the new and old fonts the same. Font numbers are small integers (currently no more than 15). Font 0 (zero) always refers to the built-in or default font. The actual fonts associated with the font numbers may be set in the MGR startup file, or changed by clients on the fly (see m_loadfont()).

m_func(mode)

Set the drawing mode. This specifies the drawing mode for all graphics and bit-blt operations. The integer mode is one of 16 possible boolean combinations of the source and destination bit patterns. Combinations of bit patterns for which there is no source bitmap, such as m_bitwrite() or m_line() use the modes shown in the middle column of the following table. Several common modes are specified for the bit patterns in which the source bitmap is relevent.

Names for bit-blt modes _
source no source comments =
B_OR B_SET default
B_COPY
B_CLEAR
B_COPYINVERTED
B_INVERT
B_XOR
B_AND

Alternately, mode may be derived with a boolean combination of B_SRC and B_DST, thus B_OR is equivalent to (B_SRC | B_DST).

m_getchar()

The macro m_getchar()is equivalent to the stdio routine getchar(), except the character is retrieved from MGR via the file pointer m_termin instead of stdin.

m_gets(buff)

A line of characters is read from MGR and placed into buff. The macro m_gets()returns NULL if the connection to MGR is severed. The macro m_gets()is equivalent to the stdio fgets() call except input is retrieved from MGR.

m_getinfo(mode)

This function requests MGR to return information back to the client program. Mode specifies one of (currently) 16 different requests. Responses are always terminated with a new line for single line responses, and with a pair of new lines for multi-line responses. Consequently, clients can request and process information requests using normal line mode processing. The following list of information requests is understood.

G_ALLMINE

Information about each window that may be written to by the client program is returned, one line of information per window, as a list of space separated items. The first two items give the location of the top left corner of the window in display coordinates. The second two items give the height and width of the window, also in display coordinates. The next field contains the last two characters of the pseudotty associated with each window. Normally the pseudotty is the same for each window reported. The next field contains the window id, which is 0 (zero) for the primary window, and the value returned by the call to m_makewindow()for the other windows (if any). The final field contains the visual status of the window, which is either C_EXPOSED ('e') if the window is completely visible, or C_OBSCURED ('o') if the window is partly or completely obscured. The window information is printed in order from front to back. A sample line might look something like: center box; c. 492 2 652 570 p6 2 o which indicates that the window at (492,2) is 652 pixels wide and 570 pixels high, has a controlling pseudotty of /dev/ttyp6, is alternate window number 2, and is at least partially obscured.

G_ALL

Information about all windows is returned, one line of information per window, as a list of space separated items. The first two items give the location of the top left corner of the window in display coordinates. The second two items give the height and width of the window, also in display coordinates. The next field contains the last two characters of the pseudotty associated with each window. Normally the pseudotty is the same for each window controlled by the same client. The next field contains the window id, which is 0 (zero) for a primary window, and the value returned by the call to m_makewindow()for alternate windows. The final field contains the visual status of the window, which is either C_EXPOSED ('e') if the window is completely visible, or C_OBSCURED ('o') if the window is partly or completely obscured. The window information for each window is printed in order from front to

back. Thus the first line returned is currently the active window. A sample line might look something like:
center box; c. 492 2 652 570 p6 0 o
Which indicates that the window at (492,2) is 652 pixels wide and 570 pixels high, has a controlling pseudotty of /dev/ttyp6, is a main window, and is at least partially obscured.

G_COORDS
A single line is returned containing the location and size of the window in display coordinates. The first pair of numbers is the position of the top left corner of the window, the second pair of numbers is the window's width and height in pixels.

G_CURSOR
A single line is returned containing the position of the character and the graphics cursor. The first pair of numbers is the current column and row in character coordinates and the second pair of numbers is the current graphicslocation in window coordinates. The graphics cursor location is reported in either absolute or relative window coordinates, depending upon the window coordinate mode setting.

G_FONT
A single line is returned which contains current font information. The first pair of numbers is the character width and height, in pixels. The next number is the font number as would be used in a call to m_font, and the final field is the ascii name of the font.

G_ID
A single line is returned containing the window's alternate window id (0 for the main window), followed by the number of windows controlled by the client program.

G_MOUSE The mouse position, in display coordinates are returned, followed by the most recent button transition, which is one of 1,-1,2,-2. The numbers 1 and 2 represent buttons one and two on the mouse respectively. The third mouse button is reserved for system use and is not accessible to client programs. A negative value means the button was released; a positive value indicates the button is still depressed.

G_MOUSE2
The mouse coordinates, in window coordinates are returned, followed by the most recent button transition, which is one of 1,-1,2,-2. The numbers 1 and 2 represent buttons one and two respectively. A negative value means the button was last released; a positive value indicates the button is still depressed. If the mouse is above or to the left of the window, a negative coordinate value is returned. In addition if the window is in relative coordinate mode, coordinate values between 0 and 999 will be reported only if the mouse is within the window.

G_STATUS
A line is returned containing a single character, either C_EXPOSED ('e'), C_OBSCURED ('o'), or C_ACTIVE ('a') depending upon whether the window is exposed but not the active window, partially or totally obscured, or exposed and the active window.

G_SYSTEM
A single line containing constant global information is returned. There are currently four fields:

1) The hostname of the machine MGR is running on, as returned by gethostname().

2) The width of the display in pixels.

3) The height of the display in pixels.

4) The size of the window borders in pixels.

G_TERMCAP
A single line is returned which contains a TERMCAP entry for MGR. The TERMCAP entry is always the same, except for lines and columns entries (li# and co#), which vary to reflect the current window size.

G_TEXT A single line containing four integers is returned with the current text region size. The first pair of numbers is the top left corner of the text region, in window coordinates, the second pair of numbers is the width and height of the text region. If no text region is defined, implying the entire window is the text region, all four numbers are returned as 0 (zero).

G_WINSIZE
A single line is returned containing the current number of columns and rows in the text region. If no text region is defined, the number of lines and columns for the entire window is returned.

G_FLAGS
A single line is returned containing a hexadecimal number representing the current window mode bits. Each mode is represented by a bit in the word. Many of the modes may be set or cleared with m_setmode() or m_clearmode(). See the discussion of m_setmode() for a detailed discussion of these flags. The meaningful mode bits are:

0x000001 The window is completely exposed.

0x000004 It is possible to use the system cut function in this window. This mode is restored by clearing the window. See m_clear().

0x000008 The window is white text on a black background.

0x000010 The window is in standout mode. Individual characters are printed in reverse.

0x000020 The window has died. If a client sees this flag, the window is about to go away.

0x000040 Expose the window upon shell output. The window will be automatically activated when the next character arrives for output on the window.

0x000080 Permit a partially or totally obscured window to update.

0x000100 Do not kill the window when the original process started in it dies. This flag may only be set from the startup file.

0x000200 Vi mode is turned on. Pushing the right mouse button sends the characters:
row column
where row and column specifies the character location the mouse is sitting on. This has the effect of aligning vi's notion of the current character position with the mouse.

0x000800 Keyboard input is refused when the window is active.

0x001000 Auto wrap mode is turned on. The character cursor automatically wraps to the beginning of the next row when it reaches the right margin of the text region.

0x002000 Overstrike mode is turned on. Characters are written to the window using the the current drawing mode, as set by m_func().

0x004000 The window is in absolute window coordinate mode.

0x010000 The system cut function snarfs complete lines only.

0x020000 The system cut function changes spaces to tabs whenever possible. Tabs are assumed to be every 8 spaces.

0x040000 The system cut function will attempt to snarf text even if errors occur.

m_go(x,y)

Move the graphics point to the window position (x,y) in the current window coordinates.

m_gotext()

The graphics point is moved to the bottom left corner of the current character cursor location.

m_halfwin(X,Y,Dwidth,Dheight)

A window is created at X,Y of size Dwidth by Dheight with no process connected to it. MGR returns the name of the file, a pseudo-tty, that must be opened in order to talk to the new window. A process which opens that pseudo-tty becomes a client program, communicating with MGR and the new window in the usual fashion. For a single process managing multiple windows, use m_newwin().

m_highlight(X,Y,Dwidth,Dheight)

MGR flashes the rectagular portion of the display starting at X,Y of size Dwidth by Dheight. This is an experimental capability and may be removed in the future.

m_incr(n)

The current character position is adjusted to the left or right n units in window coordinates. The argument n may be signed to indicate movement to the left (if negative) or to the right (if positive or unsigned). This is useful for client programs dealing with proportionally spaced text.

m_left(n)

Move the character cursor left n tenths of a character width. See also m_down()m_right()and m_up().

m_line(x1,y1,x2,y2)

Draw a line in the current window from the coordinate (x1,y1) to the coordinate (x2,y2). The line is either set, cleared or inverted as determined by the last call to m_func().

m_linecolor(mode,color)

The drawing mode and color is set for all graphics and bit-blt operations. The integer mode sets the drawing mode, in the manner of m_func(). The integer color is the index into the color lookup table for the drawing color. This command is equivalent to m_func()on a monochrome display. See also m_fcolor()and m_linecolor().

m_lineto(to,x1,y1,x2,y2)

Draw a line on the scratchpad bitmap to from the coordinate (x1,y1) to the coordinate (x2,y2). The line is either set, cleared or inverted as determined by the last call to m_func().

m_linkmenu(parent,item,child,mode)

The menus parent and child are linked together. When menu parent is popped up and item number item (starting from zero) is highlighted, sliding off to the right of the parent menu causes the child menu to pop up. When an item is chosen, MGR sends the concatenation of the action strings associated with each of the popped-up menus, from left to right (i.e. parent to child). An arbitrary tree of menus may be created by linking successive menus together in this manner. It is up to the application to indicate on the parent menu item that sliding to the right will pop up a child menu. Typically "->" is used.

The mode argument, if not zero, changes the menu options for the parent menu. The flag settings, which may be or-ed together (except for MF_CLEAR) are:

MF_SNIP

By default, when an item in a child menu is selected, the values associated

with the highlighted items for all of the ancestor menus are concatenated to the child's item value. When MF_SNIP is enabled, only the string associated with the child menu is returned.

MF_PAGE
Normally, whenever a menu is popped-up, the previously chosen item is initially highlighted. If MF_PAGE is enabled, this behavior is extended to paged menus. MGR automatically pages through a set of paged menus to highlight the currently selected item.

MF_AUTO
MGR will automatically slide to the right and pop up a child menu to highlight the previously selected item.

MF_CLEAR
Clears the mode MF_SNIP, MF_PAGE, and MF_AUTO. See also m_loadmenu(), m_selectmenu(), and m_unlinkmenu().

m_loadfont(n,name)
The MGR font whose pathname is name is downloaded into MGR, replacing the font currently located at position n. Any subsequent calls to m_font()will select the newly downloaded font. The font that used to be at position n remains available to windows that are already using it, but is unavailable for future use. The format of MGR font files is described in font.h.

m_loadmenu(n,string)
The text string is downloaded into menu position n. The first character of string is the menu delimiter character. All of the menu item strings are concatenated, followed by all of their action strings. The menu delimiter character separates all of the items and actions and terminates the list. Menus are downloaded at once, as a single entity. The macro m_selectmenu() is used to have the menu pop-up when a mouse button is pushed.

m_move(column,row)
The character cursor is moved to character location column,row , where (0,0) is the top left character position on the window, or on the current text region if one is specified (see m_textregion()).

m_movecursor(x,y)
Move the character cursor to the position (x,y) in window coordinates. This permits characters to be placed at arbitrary pixel locations, not just on character boundaries. Use m_move()to move to a row and column position.

m_movemouse(X,Y)
Move the mouse to position (X,Y) in display coordinates. Excessive use of this macro is anti-social.

m_moveprint(x,y,string)

Print string at the window coordinate (x,y). This macro is equivalent to calling m_movecursor() followed by m_printstr().

m_movewindow(X,Y)

Move the window to the display location (X,Y) in display coordinates. If the new position is too close to the edge of the display for the window to fit entirely at the requested location, the right edge or bottom of the window is truncated at the boundary of the display. An alternate window is created with the size and location indicated. The arguments X and Y specify the upper left corner of the window, Dwide and Dhigh the size. If the window is to be to fit at the requested location, its size is truncated appropriately. MGR will return a window number if the creation is successful, or a newline if the window could not be created. The newly created window is made the active window. The macro m_selectwin()is used to enable writing on the newly created window.

m_nomenu()

Deselect all menus. No menu will pop-up when the middle mouse button is pressed. This call does not delete the menu, it simply disassociates it from the button.

m_nomenu2()

Deselect all menus. No menu will pop-up when the right mouse button is pressed. This call does not delete the menu, it simply disassociates it from the button. This macro should be combined with m_nomenu() but is separate for historical reasons.

m_othersave(id,sub,name)

The bitmap contents of the window identified as id.sub is saved in the file name in bitmap format (see dump.h for a description of the bitmap format). The window id can be determined either by calling m_getinfo(G_ALL) or from the event M_ACCEPT (see m_setevent()). See also m_windowsave() and m_bitsave().

m_pagemenu(parent,child)

Connect menu child to the bottom of menu parent to permit a long menu to be paged. Mousing off the bottom of the parent menu automatically pops up the child menu, which in turn may be the parent of another menu. See also m_unpagemenu(), m_linkmenu() and m_unlinkmenu().

m_pop()

Pop the window context. The last window context saved by calling m_push() or m_pushsave() is restored. If no environments have been pushed, m_pop()is ignored.

m_popall()

Like m_pop()above, except all environments pushed since the first call to m_setup()are popped. The macro m_popall() is typically used as part of the clean up before client program termination.

m_printstr(string)

Print string on the window at the current character cursor location. This is equivalent to the stdio function printf with a %s format specified and the output directed toward the file pointer m_termout instead of stdout.

m_push(mode)

Certain parts (stack modes) of the current window environment may be moved to a stack, to be restored at a later time with m_pop()or m_popall(). Any combination of the following pieces of the window environment, called a window context, may be placed on the window stack.

P_BITMAP
All currently defined scratchpad bitmaps are moved to the stack and become undefined in the current window context.

P_CURSOR
The current character cursor and graphics point positions are saved on the stack.

P_EVENT
All currently defined events are moved to the stack and become undefined in the current window context.

P_FLAGS The window modes, as set with m_setmode()are moved to the stack. The modes revert to their default settings in the current window context.

P_FONT
The current font setting is copied to the stack. If this font is subsequently deleted, by writing over it with a different font, the original font setting is retained, even if it can no longer be accessed using m_font().

P_MENU
All downloaded menus and menu links are moved to the the stack along with the currently selected menu number. The menus become undefined in the current context.

P_MOUSE
The mouse cursor location is saved on the stack. Its current location remains the same.

P_POSITION
The window size and location is saved on the stack. The current size and location are maintained.

P_TEXT
The text region location and size are saved on the stack. The text region in the current context is reset to the entire window.

P_WINDOW
The current image contents of the window is copied to the stack. This is done without altering the current contents of the window. Stack modes are combined by or-ing them together to form a saved window context,

such as: m_push(P_MENU|P_EVENT) which will save all events, and menus but leave everything else alone. All stack modes that require client download data revert to their default settings when they are pushed. For example, after m_push(P_MENU|P_EVENT|P_MOUSE) is called, no events or menus are currently defined, but the mouse remains where it is. The defined constant P_ALL refers to all of the modes.

m_pushsave(mode)

Certain parts (stack modes) of the current window environment may be copied to a stack, to be restored at a later time with m_pop()or m_popall(). The macro m_pushsave() differs from m_push()in that downloaded data, such as menus events or scratchpad bitmaps are copied to the stack instead of moved, and thus remain in effect after the call to m_pushsave(). The current window context is thus unaffected. Any combination of the following pieces of the window environment may be copied to the window stack.

P_BITMAP
All currently defined scratchpad bitmaps are copied to the stack.

P_CURSOR
The current character and graphics cursor positions are saved on the stack.

P_EVENT
All currently defined events are copied to the stack.

P_FLAGS
The window modes, as set with m_setmode(), are copied to the stack.

P_FONT
The current font setting is copied to the stack. If this font is subsequently deleted, by writing over it with a different font, the original font setting is retained, even if it can no longer be accessed using m_font().

P_MENU
All downloaded menus and menu links are copied to the the stack along with the currently selected menu number.

P_MOUSE
The mouse cursor location is saved on the stack.

P_POSITION
The window size and location are saved on the stack.

P_TEXT
The text region location and size are saved on the stack.

P_WINDOW
The current image contents of the window is copied to the stack. Stack modes are combined by or-ing them together to form a saved window context, such as: m_push(P_MENU|P_EVENT) which will save all events, and menus but leave everything else alone.

m_put(string)

String is put into the global snarf buffer. There is one common buffer for all clients programs. The macro m_snarf() is used to retrieve the contents of the buffer. The MGR system cut function places text in this buffer, whereas the system paste function pastes text from this buffer.

m_putchar(c)

The character c is written in the window at the current character cursor location. This function is like the stdio putchar(c), only directed toward the client's window.

m_rcircle(radius)

A circle of radius radius is drawn, centered at the current graphics point. The points at the edge of the circle are set, cleared or inverted depending upon the last call to m_func(). Circles are always drawn as circles, both in absolute and in relative window coordinates. The radius is scaled based upon the average width and height of the window.

m_rellipse(radius1,radius2)

Draw an ellipse centered at the graphics point. The two radii, radius1 and radius2 specify the major and minor axis. The ellipse is either set, cleared, or inverted determined by the last call to m_func(). If the window is in relative coordinate mode, radius1 and radius2 are scaled based upon the average width and height of the window.

m_resetesc()

The MGR escape character is reset to its to default value ('\033'). This turns off the debugging mode turned on by m_setesc().

m_rfastdraw(count,data)

The next count bytes of data are sent to MGR are to be interpreted as lines drawn in fast draw mode, starting at The current graphics point. Fast draw mode permits the rapid drawing of short vectors by encoding an x,y displacement location in a single byte. The x coordinate is contained in the most significant 4 bits, the y coordinate in the least significant 4 bits. Values for x and y represent displacements from the previous location, and range from +7 to -8. A 7 is coded as 0xff, a -8 as 0x00. If both x and y are zero (i.e. 0x8080). The next coordinate is taken to be a move instead of a draw. An eight bit channel between MGR and the client program is required for fast draw mode. See also m_fastdraw().

m_right(n)

Move the character cursor right n tenths of a character width. See also m_left()m_down()and m_up().

m_scrollregion(first_row,last_row)

This sets up a text region as a VT100-like scrolling region. The entire width of the window from lines first_row to last_row inclusive becomes the text region. See also m_textregion().

m_selectmenu(n)

This macro is used to indicate menu n pops-up in response to pressing the middle mouse button. Menus are downloaded (with m_loadmenu()) first, then selected. Only one menu may be selected at a time on each button. If the button is already down when this call is made, and there is not currently a menu associated with the button, then the menu just selected pops-up immediately. This last feature may be used to pop up different menus in a context sensitive way.

m_selectmenu2(n)

This macro is used to indicate menu n pops-up in response to pressing the right mouse button. Menus are downloaded (with m_loadmenu()) first, then selected. This macro functions the same as, and should be combined with m_selectmenu() above, but exists separately for historical reasons.

m_selectwin(n)

Select alternate window n for output. Alternate windows are first created by m_newwin(). All output goes to the selected window until either m_selectwin()is called to change windows, or the selected window is destroyed. If n is 0 (zero) or the currently selected window is destroyed, the main, or original window is selected. Input from all windows is send to the client program on the same input channel. The macro m_setevent(ACTIVATE) may be used to help decide what window generated the input by associating a unique string with each window's ACTIVATE event. The selected window and the active window are specified independently. Selecting a window does not make it the active window, and creating a new window, although it is created as the active window, is not automatically selected.

m_sendme(string)

The argument string is sent back to the client process as if it was typed in at the keyboard.

m_sendto(n,string)

The message string is send to window n. A unique window identifier, n is determined with either m_setevent() using the %w option, or with m_getinfo(). In general, the window id n is the process id (pid) of the client program started by MGR when the window was created. If the target window has turned on ACCEPT with m_setevent(), string is received by the client program associated with the target window as part of the ACCEPT event. General write permissions must be disabled on the target client's pseudotty in order for the message to be received, to prevent unsuspecting shells from interpreting messages sent by unscrupulous processes as commands. See also m_broadcast().

m_setesc(c)

This macro call causes the character c to be used as the MGR escape character by the library package (instead of '\033'). This permits viewing the output stream to MGR without causing the commands to be executed.

m_setecho()

Turn on character echoing, if possible. Character echoing is normally disabled by clients to inhibit information from MGR, as from calls to m_getinfo(), from echoing on the window.

m_setevent(n,string)

An event string, string is sent to the client program by MGR upon the occurrence of the specified event n. The event string is typically read by the client program using m_gets(). Event strings are never sent in response to an event unless specifically requested by the client program. Events are one of the following types.

ACTIVATE
The window became the active window. It is at the front of the display, and is currently receiving both mouse and keyboard input.

BUTTON_1
The right mouse button was depressed. This event is sent only to the active window.

BUTTON_1U
The right mouse button was released. This event is sent only to the active window.

BUTTON_2
The middle mouse button was depressed. This event is sent only to the active window.

BUTTON_2U
The middle mouse button was released. This event is sent only to the active window.

COVERED
The window was partially or completely obscured by another window.

DEACTIVATE
The window was deactivated, it is no longer the active window.

REDRAW
The display was redrawn, either by selecting the redraw option from the system menu, or by keying LEFT-r from the keyboard. Only windows that are exposed receive the REDRAW event. The images of obscured windows are restored automatically by MGR. The client program is expected to regenerate the contents of its window in response to the REDRAW event.

RESHAPE
The window was reshaped. If the user selects the system reshape option, the RESHAPE event is sent, even if the window stays the same shape.

UNCOVERED
The window, previously obscured, was uncovered. If the window also became the active window, the UNCOVERED event is sent before the ACTIVATE event.

MOVE
The window was moved.

DESTROY
The window was destroyed. Only alternate windows (as created by m_newwin()) cause DESTROY events to be sent. If the main window is destroyed, the client program is sent a hangup signal, and its connection to MGR is severed.

ACCEPT
Messages are accepted from client programs running in other windows (see m_sendto()). The content of the message is obtained by specifying the %m parameter as part of the event string, as is fully described below.

NOTIFY
Register a name with MGR, and make this name available to client programs. This name is available to other clients, either by a call to m_getinfo(G_NOTIFY) or with the %n parameter described below. Unlike the other events, the notify string is never sent back to the client program by MGR, but is used to register a name for the window.

SNARFED
Text was put into the snarf buffer either by a client program with m_put() or by use of the system cut function.

PASTE
Text is about to arrive as a result of the system pastefunction.

Some event strings may contain substitutable parameters in the manner of printf format specifiers (i.e. %X). These parameters are applicable only to certain types of events. In any case, the % character may be forced by doubling it, as in %%. Where more than one data item replaces the format specifier, the items are separated by a space character. For the event strings BUTTON_1 and BUTTON_2, several parameters cause MGR to sweep out some object in response to mouse movement, and report back the size of the swept object when the button is released. Any one of lines, boxes, text, or rectangles may be swept out with this mechanism. Initial parameters may be associated with a sweep event by listing them as comma separated integers following the % and preceding the sweep command character. The parameters (if any) set the initial size of the object to be swept, in the same coordinate system in which the sweep extend is reported.

%r Depressing the button causes MGR to sweep out a rectangle in response to moving the mouse, in a manner similar to the system reshape function. The initial parameters set the initial width and height of the rectangle. When the button is released, the coordinates of the starting and ending points of the rectangle in response to moving the mouse, in window coordinates, are substituted for the %r.

%R Depressing the button causes MGR to sweep out a rectangle, as in %r above, only the the result is in displaycoordinates.

%b Depressing the button causes MGR to move a rectangle in response to moving the mouse, in a manner similar to the system move function. The initial parameters set the initial width and height of the rectangle to be moved. When the button is released, the current coordinates of the box's corner is returned in window coordinates, substituted for the %b.

%B Depressing the button causes MGR to move a rectangle in response to moving the mouse, in a manner similar to the system move function. The initial parameters set the initial width and height of the rectangle to be moved. When the button is released, the current coordinates of the box's corner is returned in display coordinates, substituted for the %b.

%l Depressing the button causes MGR to sweep out a line. One end of the line remains fixed at the graphics point while the other end of the line tracks the mouse position. The initial end point of the line may be specified as a displacement from the graphics point as part of the initial parameters. When the button is released, the coordinates of the starting and ending points of the line, in window coordinates, are substituted for the %l.

%t Depressing the button causes MGR to sweep out text, in a manner equivalent to the system cut function. Upon the release of the button, the %t is replaced by the starting character coordinate of the cut region, followed by character distance to the ending point in columns and lines respectively. For example, The event string sweep[%t] might return sweep[17 5 6 0], indicating the user swept out a six character word on a single line, starting on column 17, row 5. An inital size may ber specified in number of rows and number of columns. The remaining format specifiers are replaced by the information described below No sweep action is performed.

%p The %p is replaced by the current mouse coordinates, in window coordinates.

%P The %P is replaced by the current mouse coordinates, in character coordinates.

%n If the mouse cursor is over a window whose NOTIFY event is set, the text of that message is substituted for the %n.

%w If the mouse cursor is over a window whose NOTIFY event is set, the window_id of the clicked on window is substituted for the %w. This window_id may be used by m_sendto()to send the clicked-on window a message.

%S If the mouse cursor is over a window whose NOTIFY event is set, the length of that message is substituted for the %S.

The ACCEPT event is used to receive messages from other client programs. The following substituteable parameters may be used as part of the event string.

%f The window_id of message sender, as used in m_sendto(), replaces the %f.

%m The text of message sent by the other client program replaces the %m

%s The length of the message, in characters, replaces the %s. For example, a call to
center box; c. m_setevent(ACCEPT,"Message from (%f), (%s) long is: %m")
might cause MGR to return
center box; c. Message from (3214), (2) long is: HI
after the client program whose window id is 3214 uses m_sendto()to send the message "HI".

%p As with the BUTTON events above, %p is replaced by the current mouse position in window coordinates.

%P As with the BUTTON events above, %P is replaced by the current mouse position in character coordinates. For the SNARFED event string, the following substitution parameters are recognized.

%f The window id of the window filling the snarf buffer replaces the %f.

%c The current length of the snarf buffer, in characters, replaces the %c.

%C The contents of the snarf buffer replaces the %C. At present, only the first 250 characters of the snarf buffer may be returned via the %C parameter. Use m_snarf()to read the entire buffer.

The PASTE event string, recognizes the %c specifier as described under SNARFED above.

m_setmode(mode)

Various window modes may be set or cleared (see m_clearmode()) independently. These modes are:

M_STANDOUT
The window is put in standout mode. All characters are written with their foreground and background colors reversed.

M_WOBThe sense of white and black is reversed for the entire window, not just for characters as is M_STANDOUT.

M_AUTOEXPOSE
The next character to be typed on the window causes it to automatically become the active window.

M_BACKGROUND
Output goes to the window even if it is partially or totally obscured. The data in exposed portions of the window is seen immediately. Data in covered portions of the window is saved by MGR and restored when the covered portions are exposed.

M_NOINPUT
Keyboard input is prohibited. All input from the keyboard is held buffered by MGR until either M_NOINPUT is cleared, or a different window is made the active window. In the latter case the input goes to the newly activated window. This flag is automatically turned off when the user activates the window. This feature is for client programs that want one of their windows to come to the front just long enough to notify the user of some event, but do not want to accidently intercept keyboard input while the user is merrily typing to some other client.

M_NOWRAP
The character cursor does not automatically jump to the left edge of the next line as it reaches the right edge of its text region. After the right margin is passed, the cursor and any subsequent text disappear past the right edge of the window.

M_OVERSTRIKE
Text is written to the window with the mode specified by m_func() instead of the normal copy mode. In copy mode, the characters completely obliterate their destination instead of combining with it.

M_ABS
The window is set to absolute coordinate mode. The upper left edge of the window, just inside the border is at position (0,0). All other locations are measured relative to that corner in pixels.

M_ACTIVATE
The window is made the active window, pops to the front of the display, and obtains all keyboard and mouse input.

M_SNARFLINES
The system cut function only cuts entire lines. If any text on a line is swept out, the entire line of text is included.

M_SNARFTABS
The system cut function attempts to turn white space into a minimal combination of spaces and tabs. Tab are set at every 8 columns.

M_SNARFHARD
The system cut function attempts to cut text even if the window contents have been corrupted. Unidentifyable characters are returned as C_NO-CHAR ('?').

M_STACK
Any events pushed on the window stack when this flag is set will be sent in addition to any currently active events. This setting is useful for filters which need to receive events, yet still permit clients running under them to receive events as well.

m_setnoecho()
Character echoing to the window is disabled if possible. Character echoing is normally disabled by clients to inhibit information from MGR, as from calls to m_getinfo() from echoing on the window.

m_setnoraw()
Normal terminal input processing is in effect. Input is buffered by lines, and all normal line editting and keyboard interrupt generation is in effect.

m_setraw()
Every character is available as entered, no input processing is done. This is typically called raw mode. Raw mode is not always available, in which case the macro call is ignored.

m_shapewindow(X,Y,Dwidth,Dheight)
The window is reshaped to position (X,Y) and with size Dwidth by Dheight. As only the active window may be reshaped, m_shapewindow()activates the window if it is not already active. The new size of the window is not guaranteed; the width or height may be truncated to the right or bottom edges of the display. The macro m_getinfo()can be used to determine the actual window size.

m_size(columns,rows)
The size of the window is changed so that it fits exactly columns by rows of characters in the current font. The window may be truncated at the right or bottom edge of the display if it is too large to fit on the display at its current position.

m_sizeall(X,Y,columns,rows)
The window is reshaped to position (X,Y) on the display, and resized to fit columns and rows of text. As only the active window may be reshaped, m_shapewindow() activates the window if it is not already active. The new size of the window is not guaranteed; the width or height may be truncated to the right or bottom edges of the display. The macro m_getinfo()can be used to determine the actual window size.

m_snarf()
The application is sent the contents of the global snarf buffer, if any, as specified by the last call by a client programs call to m_put() or by use of the system cut function.

m_sleep()
This call causes MGR to suspend the processing of characters to the

window. After a chunk of output for all other windows has been processed, output processing resumes. This does not normally take very long, making m_sleep() almost a no-op.

m_standend()

Inverse video mode as set by m_standout() is turned off. This is exactly equivalent to m_clearmode(M_STANDOUT).

m_standout()

Inverse video mode is turned on. This is exactly equivalent to m_setmode(M_STANDOUT). The color of the characters and their backgrounds are interchanged.

m_stringto(to,x_dst,y_dst,string)

The text string is printed starting at the location (x_dst,y_dst) on scratchpad bitmap to. The text is clipped to fit in the bitmap, and no special command processing is done on string. If to is 0 (zero), The text is printed on the window, but text region boundaries are ignored. This is the only way to get text into a window outside of the text region.

m_textregion(x,y,wide,high)

A subregion within the current window starting at (x,y) and of size wide by high is defined within which all text is restricted. All functions and information that deals in character coordinates views the text region as if it was the entire window. As soon as the text region is defined, the character cursor is moved to row and column (0,0), which is now located at the point (x,y). Graphics output is not affected by text regions.

m_textreset()

The text region (defined by a call to m_textregion()) is reset to be the entire window. This is the default setting.

m_unlinkmenu(parent,item)

The menu link associating a child menu with the menu parent at item (counting from zero) is removed (see also m_linkmenu()). This function does not change the menus, only their connections.

m_unpagemenu(parent)

The link associating the menu parent with a child menu is removed. See also m_pagemenu().

m_up(n)

Move the character cursor up n tenths of a character height. This may cause the window to scroll down. See also m_left()m_right()and m_down().

m_whatsat(X,Y)

MGR returns to the client program a line indicating what is at the display coordinates (X,Y). If that location is occupied by a window, a line containing the window's controlling terminal, alternate window number, and window id is returned in a space separated list. If the location (X,Y) is not in a window, MGR returns a newline.

m_windowsave(name)

The current image contents of the window is saved in the file name on the MGR-host machine in MGR bitmap format. File names beginning with "./" are evaluated relative to the current directory when MGR was started. See also m_othersave() and m_bitsave().

Functions

The functions listed below have packaged common sequences of macro calls together to provide a slightly higher level of interface than the macros alone. They are still low level, and have no pretense of completeness. Except where noted, all of the functions return a value greater than zero on success, and a value less than zero upon failure. The functions fail only if they read an unexpected value from MGR. Client programs may use the function m_lastline()in an attempt to determine what input caused the failure. Those functions which expect data from MGR automatically flush any pending output before reading, and unless the M_MODEOK flag is set, attempt to turn off character echoing to prevent data returned by MGR from echoing back on the window.

int
get_all(list)
struct window_data *list;

The current position size and status of all windows on the display is returned in list. The number of in windows on the display is returned. List should be large enough to hold a status entry for each window. The window_data structure is defined in term.h.

int
get_client(list)
struct window_data *list;

The current position size and status of the client programs main and alternate windows is returned in list. The number of windows owned by the client program is returned. List should be large enough to hold a status entry for each window. The window_data structure is defined in term.h.

int
get_colrow(columns,rows)
int *columns, *rows;

The number of columns and rows in the current text region is returned in columns and rows respectively. For any NULL argument, no value is returned.

int
get_cursor(column,row)
int *column, *row;

The current character cursor position is placed in column and row. For any NULL argument, no value is returned.

```
int
get_eachclientwin( windatap)
struct window_data *windatap;
```

Get the window parameters for each window in the current window set, one window at a time. This function returns 1 if window_data structure has been filled, 0 otherwise. It is important to call get_eachcleintwin() in a tight loop that doesn't exit until it returns 0, so that all the data is picked up. This function is preferred to get_client() because you don't need to know the maximum number of windows you are likely to see.

```
int
get_eachwin( windatap)
struct window_data *windatap;
```

Get the window parameters for all the windows, one window at a time. This function returns 1 if window_data structure has been filled, 0 otherwise. It is important to call get_eachwin() in a tight loop that doesn't exit until it returns 0, so that all the data is picked up. This function is preferred to get_all() because you don't need to know the maximum number of windows you are likely to see.

```
int
get_font( wide,high)
int *wide, *high;
```

The character size of the current font, in pixels is placed in wide and high. For any NULL argument, no value is returned. The function returns the current font number, as would be used in a call to m_font().

```
int
get_mouse( x,y)
int *x, *y;
```

The current mouse position, in window coordinates, is placed in x and y. For any NULL argument, no value is returned. The function returns the current mouse button state, which is in the range of -2 to +2 upon success, a value less than -2 upon failure. See m_getinfo(G_MOUSE) for a discussion of the return values.

```
int
get_param( host,xmax,ymax,border)
char *host;
int *xmax, *ymax, *border;
```

The MGR-host, display size (in pixels) and window border size (in pixels) is placed in the arguments host,xmax,ymax, and border. For any NULL argument, no value is returned.

```
int
get_size( X,Y,Dwidth,Dheight)
int *X, *Y, *Dwidth, *Dheight;
```

The position of the window on the display, in display coordinates is placed into X,Y,Dwidth and Dheight. For any NULL argument, no value is returned.

```
char *
get_termcap()
```

A string containing a TERMCAP entry, suitable for placing into the TERMCAP environment variable is returned. The function get_termcap() returns NULL upon failure.

```
int
is_active()
```

The function is_active() returns TRUE if the window is the active window.

```
void
menu_load( n,count,text)
int n;
int count;
struct menu_entry *text;
```

A menu is downloaded to MGR at position n. The integer count is the number of menu items to be down-loaded, and text is an array of menu item/value pairs. The structure menu_entry is defined in term.h.

```
int
m_bitfile( to,name,widep,highp)
int to;
char *name;
int *widep, *highp;
```

Given a bitmap id, to and an icon name, have MGR load that icon into that scratchpad bitmap, returning the icon width and height, in pixels, via the given integer pointers. Return a positive number if successful. If the icon is not loaded, set the width and height values to 0 and return 0. This function is identical to m_bitfromfile()plus the needed interception of the line returned from MGR.

```
void
m_bitload( x,y,wide,high,data)
int x,y;
int wide,high;
register char *data;
```

The bitmap image pointed at by data is down-loaded to the window at position (x,y) in window coordinates. It is up to the client program to insure an 8 bit channel exists between the client and MGR. The integers wide and high specify the size of the bitmap in pixels.

char *
m_lastline()

The last input from MGR to a library function is returned. The data is kept in a static buffer which is overwritten at each request.

int
m_makewindow(X,Y,Dwidth,Dheight)
int X, Y, Dwidth, Dheight;

An alternate window is created as the active window, at display coordinates (X,Y) and of size Dwidth by Dheight pixels. If the window is too big to fit on the display, its width and height are truncated. The alternate window's window-id is returned if the window was created successfully. The macro m_selectwin() is used to write on the newly created window.

int
m_setup(mode)
int mode;

This function initializes the library. It must be called before any other function or macro. The argument mode is one or more of the flags M_FLUSH, M_DEBUG, or M_MODEOK or-ed together. If M_FLUSH is present, all macros and function flush output to MGR after each macro call. This is slightly less efficient than letting the client program flush the data (see m_flush()) but prevent inadvertent buffering problems. The M_DEBUG flag forces the macro package to read and write from stdin and stdout respectively. Normally /dev/tty is opened for reading and writing to permit standard input or output redirection while still maintaining a connection to MGR. If /dev/tty can not be opened, as would be the case for clients invoked through rsh, the M_DEBUG flag is turned on, and stdin and stdout are used instead. The M_MODEOK flag instructs those functions which expect data from MGR to assume the terminal modes are set appropriately. Otherwise, the functions attempt to turn off character echoing and turn on line mode before fetching data from MGR. The functions m_ttyset() and m_ttyreset() can be used to set and reset the terminal modes. The function m_setup() returns its argument, with the M_DEBUG flag or-ed in if /dev/tty can not be opened.

void
m_ttyreset()

The terminal modes are restored to their state just prior to the last call to m_ttyset(). Calls to m_ttyset() and m_ttyreset() may be stacked up to ten levels.

int
m_ttyset()

The terminal is set in a state suitable for data exchange with MGR. Character echoing is turned off, and line processing mode is enabled. The function returns zero (0) if it successfully retrieves the terminal modes, -1 otherwise.

Using the Library

Clients using The C Interface Library should specify:

```
#include "term.h"
```

which also includes <stdio.h>

if it has not already been included. Programs are compiled either with

```
cc -o foo foo.c term.o -I$(lib)
```

where $(lib) is the MGR include directory or simply

```
cc -o foo foo.c -lmgr
```

if the library is installed in a standard location. The file term.o contains the functions listed in the last section. Several compile time options are available to the client program using the library. Normally, the library setup routine, m_setup() attempts to open /dev/tty to communicate with MGR. Client programs may define the C preprocessor symbols M_DEVICEIN or M_DEVICEOUT to override the selection of /dev/tty for input or output respectively. After each macro call, the flush flag M_FLUSH is tested to see if output should be flushed to MGR. If the C preprocessor symbol M_NOFLUSH is defined, before the client program includes term.h, The flush flag is never tested, and it becomes the responsibility of the client program to insure output is flushed at the appropriate times.

Several external variables maintained by the library are accessible to client programs. The stdio FILE pointers m_termin and m_termout are used for directing output to, and receiving input from MGR. These file pointers are initialized in m_setup(), and may be changed by client programs after m_setup() is called. The integer m_flags contains the current library mode settings, as returned by m_setup(). The M_MODEOK and M_FLUSH bits of m_flags may also be changed at any time after m_setup() is called. The integer m_envcount contains the current window context depth, and is used by m_popall()to pop the appropriate number of contexts. M_envcount should not be changed by client programs. Finally, the character m_menuchar defines the menu separator character used by menu_load(). This character, set to '\005' by the library package, can be changed to any character that will not appear as part of a menu
item or action.

absolute coordinates
: Absolute coordinates is a coordinate system used in windows measured in units of pixels.

active window
: The active window is the window logically in the front of the display, which is receiving keyboard and mouse input.

alternate window
: An alternate window is an additional window created by a client program that shares the communication channel with the main window.

bitmap
: A bitmap is a rectangular array of bits, or pixels if the bitmap is currently on the display.

channel
: The channel is the bidirectional byte stream connecting MGR with the client program. The channel is usually the program's standard input and output.

character coordinates
: Character coordinates is the coordinate system used in windows for counting characters in columns and rows.

character cursor
: The character cursor is the location in the window where the next character will be placed. The cursor location is always highlighted on the active window.

child menu
: A child menu is the menu that will pop up when the user slides off to the right of a popped up parent menu.

client program
: A client program is any Unix command that is running in an MGR window. Client programs may be existing programs, as might be found in /bin or new applications written specifically to take advantage of the windowing environment.

client-host
: The client-host is the computer that the client program is running on. It is often the same as the MGR-host machine, but it does not need to be.

display coordinates
: Display coordinates is a coordinate system used to measure pixels on the display. The top left corner of the display is at (0,0), with x increasing to the right, and y increasing down.

exposed
: A window is exposed if it is entirely visible.

graphics point
: The graphics point is a location on the window, measured in the prevailing window coordinate system, that may serve as a reference location or origin for many of the graphics operations.

listener
: A listener is a window that has turned on the ACCEPT event and is willing to receive messages from other client programs.

main window
: A client program's main window is the window the program was started in. The client program may create and destroy alternate windows, but not its main window. If the user destroys a client program's main window, the connection to MGR is severed, and the client program receives a hangup
signal.

MGR-host
: The MGR-host is the computer that MGR is running on.

mouse cursor
: The mouse cursor is a small bitmap or icon that tracks the movement of the mouse on the display. Normally the mouse cursor is a small arrow pointing north west.

obscured
: A window is obscured when it is partially or totally covered by another window.

parent menu
: A menu is called a parent menu when sliding off to the right of a selected item causes another menu or child menu to pop up.

relative coordinates
: Relative coordinates is a coordinate system for windows where a single unit represents one thousandth of the way across (or down) the window.

scratchpad bitmap
: A scratchpad bitmap is a chunk of memory, allocated by MGR for use by a client program, that may hold a bitmap image that is not on the display.

snarf buffer
: The snarf buffer is a buffer maintained by MGR of arbitrary size that is accessible by all client programs.

target window
: A target window is a window that is to receive a message from another window.

text region
: A text region is the part of the window in which character text and the functions that operate on characters work. The text region may be the entire window (the default) or a rectangular subregion within the window.

window border
: The window border is a thin black border around every window that separates the window from the rest of the display.

window context
: A window context contains the values for one or more aspects of the window's state. Window context a may be saved on a stack, and then restored at some later time.

window id
: A window id is a unique number assigned to every window that may be used as an identifier when sending a message to the window. Window ids have two parts, the first part is the process number of the client program that was started when the window was created, the second part is the windows alternate window number, or zero for a main window.

Sample Client Program

This program, called close, closes, or iconifies a window. It is not a terribly useful application in its own right, but it does exercise several of the library calls. When close starts up, it makes the window smaller, moves it toward the top of the display, then writes the word "closed" in it. If the window is covered by another window, it changes the word "closed" to "hidden", then flashes its window every 15 seconds as long as the window is covered. If the window is then uncovered, the word "hidden" gets changed back to "closed". Activating the window causes close to restore the window's original shape and contents, then exit.

```
/* iconify a MGR window */

#include <signal.h>
#include "term.h"

#define TIME 15 /* time interval for reminder */
#define CONTEXT P_POSITION | P_WINDOW | P_FLAGS | P_EVENT | P_CURSOR

static char line[80]; /* event input buffer */

main()
{
int clean(), timer(); /* interrupt routines */
int x,y; /* window position on display */
char *msg = "closed"; /* closed window "icon" */

/* setup the window environment */

m_setup(M_FLUSH); /* setup i/o, turn on flushing */
m_push(CONTEXT); /* save current window context */
m_setmode(M_NOWRAP); /* don't auto-wrap at right margin */
m_ttyset(); /* set up tty modes */

/* catch the appropriate signals */

signal(SIGTERM,clean); /* in case we get terminated */
signal(SIGALRM,timer); /* for the reminder service */
```

```
/* iconify the window */

get_size(&x,&y,0,0); /* fetch window coordinates */
m_sizeall(x,10,strlen(msg),1);/* move and resize window */
m_clear(); /* clear the window */
m_printstr(msg); /* print message in the window */

/* catch events */

m_setevent(ACTIVATE, "A\r"); /* window is now the active window */
m_setevent(COVERED, "C\r"); /* covered by another window */
m_setevent(UNCOVERED,"U\r"); /* completely visible */
m_setevent(REDRAW, "R\r"); /* redraw requested */

m_clearmode(M_ACTIVATE); /* bury the window */

/* wait for an event */

while(m_gets(line) != NULL) /* read a line from MGR */
switch (*line) {
case 'A': /* window is activated */
clean(); /* clean up and exit */
break;
case 'C': /* window is covered */
m_clear();
m_printstr("hidden");
alarm(TIME); /* turn on reminder */
break;
case 'R': /* system 'redraw' */
case 'U': /* window is uncovered */
m_clear();
m_printstr(msg);
alarm(0); /* turn off reminder */
break;
case 'T': /* send reminder */
m_setmode(M_WOB); /* highlight window */
m_bell(); /* ring the bell */
sleep(1);
alarm(TIME); /* reset reminder timer */
m_clearmode(M_WOB); /* reset window highlighting */
break;
}
}

clean() /* clean up and exit */
{
m_ttyreset(); /* reset tty modes */
m_popall(); /* restore window context */
exit(0);
}

timer() /* called at reminder timeout */
{
m_sendme("T\r"); /* send timeout message */
}
```

Macros and Functions by Category

Macro and Function Index

These are the pages where macros and functions are referenced. The italic page numbers are the defining references.

* The routines marked with a dagger (|) are functions, the other routines are macros. The page number on which the macro or function is defined is printed in bold face after the name. Unfortunately, I haven't got round to indexing this document in this fashion as yet.

Index

Table of Contents