999-300-340
Issue 1

630 MTG
Software Reference Manual

TRADEMARKS

The following is a listing of the trademarks that are used in this manual:

- MC68000 — Trademark of Motorola, Inc.
- UNIX — Registered trademark of AT&T

NOTICE

The information in this document is subject to change without notice. AT&T assumes no responsibility for any errors that may appear in this document.

NOTICE

The information in this document is subject to change without notice. AT&T assumes no responsibility for any errors that may appear in this document.

INTRODUCTION

This reference manual contains manual pages for the AT&T 630 MTG Software Development Package. It is intended that this manual will be used in conjunction with the *630 MTG Software Development Guide*. The *630 MTG Software Development Guide* contains tutorial descriptions of many of the commands and library subroutines discussed in this reference manual.

This manual contains a table of contents and a permuted index to help you in locating a certain manual page.

> The table of contents lists the manual pages by the categories (or sections) that they are found in this manual. Within each category the manual page is listed in alphabetical order with a short description about the manual page.
>
> The permuted index is divided into three columns. The middle column is used to search for a key word or phrase. The right column will then contain the name of the manual page that contains that command. The left column contains additional useful information about the command.

The manual pages are divided into four sections, with Section 3 containing sub-classes. Note that the section numbers correspond to the section numbering scheme of UNIX* System reference manuals.

---

* Registered trademark of AT&T

Since Section 2 is not pertinent to the 630 MTG, it is omitted.

1. Commands and Application Programs
3. Subroutines:
   - 3L. Host Resident General Library Routines
   - 3R. ROM Resident General Library Routines
   - 3R/3L. Combination of 3L and 3R Routines
   - 3M. Host Resident Mathematical Library Routines
4. File Formats
5. Miscellaneous Facilities

Section 1 (*Commands and Application Programs*) describes commands intended to be accessed directly by the user or by a command language program.

Section 3 (*Subroutines*) describes library subroutines which can be called by application programs which execute in the terminal. To access 3M routines, include **-lm** on the *dmdcc* command line. For example:

```
dmdcc prog.c -lm
```

3L and 3R libraries are automatically searched by dmdcc and do not have to be explicitly included on the dmdcc command line.

Section 4 (*File Formats*) documents the structure of particular kinds of files; for example, the format of font files is given in font(4).

Section 5 (*Miscellaneous Facilities*) contains a variety of things. Included are descriptions of math functions and machine dependent values.

Each section contains a number of independent entries (commonly referred to as "manual pages"). The name of the manual page appears in the upper corners of the page. All manual pages within each section are in alphabetical order. Some manual pages may describe several routines, commands, etc.

Manual pages use a common format. Parts of a manual page that do not apply to a specific command may be omitted. The parts of the format are described as follows:

The **NAME** gives the name(s) of the command and briefly states the purpose of the command.

The **SYNOPSIS** summarizes the use of the command (program). This part uses special typesetting characteristics of the command to denote:

Boldface strings are literals and are to be typed just as they appear.

Italic strings usually represent substitutable arguments and program names found on other manual pages. If the string is underlined, the item is typed just as it appears.

Square brackets **[ ]** around an argument indicate that the argument is optional. When an argument is given as **name** or **file**, it always refers to a *file* name.

Ellipses (...) show that the previous argument may be repeated.

The **DESCRIPTION** part discusses the subject.

The **EXAMPLES** part gives examples of usage, where appropriate.

The **FILES** part gives the file(s) associated with the command.

The **SEE ALSO** part gives references to other related information.

The **DIAGNOSTICS** part describes the diagnostic indications that may be produced. Self-explanatory messages will not be described.

The **WARNINGS** part tells of potential pitfalls.

The **BUGS** part gives known bugs and occasional deficiencies. The recommended repair may be described.

TABLE OF CONTENTS

1. Commands and Application Programs

3. Subroutines

4. Files Format

5. Miscellaneous Routines

PERMUTED INDEX

NAME

dmdcat - send files to a 630 MTG connected printer.

SYNOPSIS

dmdcat [**-s**] [**-b**] [**-v**] [**-u**] [**-t**] [**-e**] [file...]

DESCRIPTION

The *dmdcat* command is intended to be used to send files to a printer connected to the Printer port of the 630 MTG terminal. *Dmdcat* will send the concatenation of files specified on its command line, or the standard input if no files are specified.

The data is sent to the terminal preceded by a *Printer-On* Request escape sequence, and is terminated by a *Printer-Off* request escape sequence. The *Printer-On* request escape sequence commands the terminal emulator to start sending incoming data to the printer if the printer is available. The *Printer-Off* request escape sequence tells the terminal emulator to stop the sending. The escape sequences sent are:

Printer On - ESC[?5;1i

Printer Off - ESC[?4i

If the **-s** is present, *dmdcat* uses this set of escape sequences:

Printer On (no screen) - ESC[?5;2i

Printer Off (no screen) - ESC[4i

which tells the terminal emulator to start/stop sending incoming data to the printer (if the printer is available) as before but not to display this data on the screen.

The terminal responds with:

ESC[?**ps**i where:

ps=0 indicates printer was not granted

ps=1 indicates printer was granted.

If the printer was not granted, or if the terminal does not respond, *dmdcat* displays a message and aborts.

The second option, **-b**, strips backspaces from the output of *dmdcat*. If backspaces result in two or more characters appearing in the same place, only the last character read is output. This means that the printed output appears exactly as it appears on the 630 MTG screen, without bold and underline. This option is useful for printers which either cannot process backspaces or are slow in processing backspaces.

Dmdcat is a shell program that calls *cat*(1) and will pass the options **-u**, **-v**, **-t**, and **-e** to *cat*(1).

FILES

$DMD/lib/dmdgetpr reads terminal response

SEE ALSO

cat(1), col(1) in the *UNIX System V User Reference Manual.*
630 MTG Terminal User's Guide.

DIAGNOSTICS

Dmdcat uses the *col*(1) command to strip backspaces with the **-b** option. *Col*(1) is not available on all UNIX systems. The **-b** option will give an error message if it cannot locate the *col*(1) command.

Dmdcat only works if it is executed from the default 630 MTG terminal emulator or any terminal emulator that supports the escape sequence sets described above.

NAME

dmdcc – 630 MTG C compiler

SYNOPSIS

dmdcc [options **]** file ...

DESCRIPTION

The *dmdcc* command is the 630 MTG C compiler. Any software to be downloaded into the 630 MTG must be compiled using this command.

Dmdcc works in a similar manner to other compiler (cc) commands but is enhanced to call *mc68cpp* and *mc68ld* with special arguments for the 630 MTG development environment. In particular, the *dmdcc* command defines the variable **DMD630**, it sets the include search path to $DMD/include, it sets the library search path to $DMD/lib, it includes the standard 630 MTG libraries, it links in the 630 MTG C run-time start-up routine crtm.o, and it tells *mc68ld* to retain relocation information so the resulting executable file can be relocated before download into the 630 MTG.

The exact arguments passed to *mc68cpp* and *mc68ld* can be viewed by including the -# debugging argument on the *dmdcc* command line.

The *dmdcc* utility accepts three types of arguments:

.c
.s
.o

Arguments whose names end with **.c** are the C source programs, and those with **.s** are the assembly programs. They are compiled/assembled, and each object program whose name is that of the source with **.o** substituted for **.c** or **.s** is left in the file. The **.o** file is normally deleted if a single C program is compiled and link-edited all at one time.

The following flags are interpreted by *dmdcc*. See *mc68cpp*(**1**), *mc68as*(1) and *mc68ld*(1) for other useful flags.

-c Suppress the link-editing phase of the compilation, and force an object file to be produced even if only one program is compiled.

-g Flag to the compiler to produce additional information needed for the use of *dmdpi*(1).

-O Invoke an object-code optimizer. The optimizer will move, merge, and delete code; this option should not be used if it is expected that compiled code may be debugged with *dmdpi*(1).

-W*c*,arg1[,arg2...]
Hand off the argument[s] to pass *c* where *c* is one of [**p02al**] indicating preprocessor, compiler, optimizer, assembler, or link editor, respectively. For example:

-Wa,-m

invokes the *m4* macro preprocessor on the input to the assembler.

-S Compile the named C programs, and leave the assembler-language output on corresponding files suffixed **.s**.

-P Run only the macro preprocessor on the named C programs, and leave the output on corresponding files suffixed **.i**.

-E Same as the **-P** option except the output is directed to the standard output. This allows the preprocessor to be used as a filter for any other compiler.

-# Debug flag. Show the command lines passed to *mc68cpp*, *mc68ccom*, *mc68as* and *mc68ld*.

-x Turn off special processing for the 630 MTG environment. This argument should not be used when compiling programs to be downloaded into the 630 MTG.

-Z *n* Allocate *n* bytes of stack for process. If not specified the default is 2048, Note that stack size can be overridden at download time with the *dmdld* -Z option. If *n* is specified smaller than 2048, it is defaulted to 2048.

Other arguments are taken to be either C preprocessor or link-editor flag arguments, or C-compatible object programs, typically produced by an earlier *dmdcc* run, or perhaps libraries of C-compatible routines. These programs, together with the results of any compilations specified, are link-edited (in the order given) to produce an executable program with name **dmda.out** unless the **-o** option of the link-editor is used.

The *dmdcc* command expects the **DMD** shell variable to be set and exported in the user's environment. This variable must point to the "root" directory of the 630 MTG software node.

Dmdcc tags the downloadable ouput file with a programming environment identification number (PEID) which is used by *dmdld* prior to download to verify compatibility with terminal firmware. The PEID is determined by *dmdcc* from the firmware routine linkages included in the process. Every firmware routine called by the process causes a linkage to the routine to be retrieved from archive and included in the load module. Each of these linkage routines is tagged with a PEID related to the firmware version that supports the called firmware routine. The PEID associated with the latest level of firmware required to support those linkages retrieved from archive is tagged to the load module. Compatibility with earlier firmware releases is maintained as long as new firmware functions not supported by older firmware are not used in the program.

FILES

file.c	input file
file.o	object file
file.s	assembly language file
dmda.out	link-edited output
/usr/tmp/mc68?	temporary file
$DMD/lib/mc68cpp	preprocessor
$DMD/lib/mc68ccom	compiler
$DMD/lib/mc68optim	optimizer
$DMD/bin/mc68as	assembler

$DMD/bin/mc68ld link loader

SEE ALSO

dmdpi(1), mc68as(1), mc68cpp(1), mc68ld(1).
m4(1) in the *UNIX System V Programmer's Reference Manual.*

The C Programming Language, by Kernighan, B. W., and Ritchie, D. M., Prentice-Hall, 1978.
Programming in C–A Tutorial by Kernighan, B. W.
C Reference Manual by Ritchie, D. M.

DIAGNOSTICS

The diagnostics produced by the C compiler are sometimes cryptic. Occasional messages may be produced by the assembler or link-editor.

NAME

dmddemo - demonstrations available on the 630 MTG

SYNOPSIS

dmddemo [options]

DESCRIPTION

The *dmddemos* are graphical demonstration programs that run on the 630 MTG. All programs will become local unless they are downloaded into a window which is the last window connected to a host. Exiting each program can be done by typing 'q' when that window is current or by deleting the window. Typing *dmddemo* with no options gives you a list of demonstrations available.

Options (demos) available are:

ball A bouncing ball

bounce Ricocheting lines. Button 1 controls speed.

clock Displays face of clock with moving hands in addition to a digital time display.

doodle An interactive doodler. Button 1 draws and button 2 erases.

rose Rotates trigonometric figures to produce various flower shapes.

star Interactive drawing. Button 1 draws and button 2 erases.

NAME

dmdld – 630 MTG application bootstrap loader

SYNOPSIS

dmdld [options **]** file **[** application - arguments **]**

DESCRIPTION

The *dmdld* program downloads the named *file* from the host, for execution in the 630 MTG terminal's window connected to its standard output. It can also be used to invoke cached applications.

The *dmdld* program works in *layers(1)* and non-layers environments. In *layers(1)*, the download runs on top of the existing *xt* error-correcting protocol. In non-layers, *dmdld* temporarily puts the 630 MTG terminal into *xt* protocol, and mimics that protocol from its side, thus insuring an error-free download.

Dmdld first asks the terminal if there is a cached application of name *file* already in the terminal. The inquiry uses the filename clipped from any pathname prefix. If such an application exists and is available, that application will be booted in the window without going through the download sequence.

If a cached application of inquired filename does not exist in the terminal or is not available (see *cache(3L)* for reasons), *dmdld* will attempt to download *file* from the host. Files to be downloaded by *dmdld* must be 630 object files created with *dmdcc(1)*.

During compilation, *dmdcc(1)* looks for the programming environment identification (PEID) number of all the library functions link-loaded, and assumes the highest number as the PEID of the application. During the download initialization, *dmdld* will send this number to the terminal. If the terminal's firmware version does not support this PEID - in other words, the firmware does not have some new library functions used by the application, the download will be aborted. The argument flag **-f** will override this checking, but the sanity of the application (after being downloaded and running) cannot be guaranteed.

The optional *application-arguments* are also sent to the terminal in order to initialize the parameters *argc* and *argv* of the function *main* of the application.

During the download, the 630 mouse cursor will turn into a "coffee cup" and the progress of the download is shown by a gradual filling of the window with inverse video. The code to be downloaded is relocated on-the-fly by *dmdld* to the memory area allocated for it by the terminal. If the download succeeds, the application will take over the window and start execution.

The following options are supported by *dmdld*:

-d causes a printout of the download information on the diagnostic output (standard error).

-p prints non-layers protocol statistics on the diagnostic output (standard error). Note that this option forces the **-d** option. In *layers(1)*,

this option is the same as the **-d** option.

-z loads the process but does not run it. The process can then be started using *dmdpi*(1). This option works only under *layers*(1).

-f forces the download even if the programming environment identification number of the application is not supported by the terminal it is downloaded into.

-n is a null option, and is ignored by *dmdld*. It is used by processes that want to fork *dmdld* with a variable argument option.

-Z *n* overrides the inherent stack size of the download application and sets it to *n* bytes. *N* must be greater than or equal to 2048, or else stack size is defaulted to 2048. Inherent stack size of the download application is specified through the -Z option of *dmdcc(1)* (or defaulted to 2048).

-T loads an absolute file (already link-loaded to a fixed address) such as a new version of the terminal's firmware into the terminal's RAM space. This download is called a takeover (overlay) download because it will close all physical ports except for the one running *dmdld*. The whole terminal screen will turn blank, and will be gradually filled up with inverse video representing the code being downloaded. When the download is finished, execution will begin at the first address of *file*. Relocation will be done by *dmdld* only if the first address of *file* is lower than the first available RAM address of the terminal. This is necessary because low-addressed RAM is used to store the terminal's system tables and variables, and overwriting them with the downloaded data may put the terminal into undetermined states. The **-N** option can be used instead, if the relocation possibility is not wanted.

-N loads an absolute file into the terminal's RAM space. The difference between the **-T** and **-N** options is that the **-N** does not relocate the absolute file. Therefore the absolute file can be generated without relocation information.

The **-T** and **-N** options only work in the non-layers environment, and are exclusive of each other.

The environment variable **JPATH** is the analog of the shell's **PATH** variable to define a set of directories in which to search for *file*.

NOTE: Standard error should be redirected when using the **-d** or **-p** options.

EXAMPLE

Invoking the terminal resident **PF Edit** application using *dmdld*:

```
dmdld "PF Edit"
```

Invoking a *dmdcc(1)* compiled application:

```
dmdld dmda.out
```

Invoking a *dmdcc(1)* compiled application with the **-d** flag, redirecting standard error to *temp*:

```
dmdld -d dmda.out 2>temp
```

Invoking a *dmdcc(1)* compiled application with *application-argument*:

```
dmdld $DMD/lib/demolib/clock "`date`"
```

SEE ALSO

cache(3L), dmdcc(1), dmdpi(1), jx(1).
layers(1) in the *UNIX System V Release 3 User's Reference Manual.*
layers(1) in the *5620 Dot-Mapped Display Reference Manual.*

DIAGNOSTICS

The error message "*dmdld: ... is not compatible with terminal*" means that the application the user attempts to download cannot execute safely in the terminal because it calls library routines which do not exist in the terminal's firmware version (i.e. the programming environment ID of the application is "newer" than the one supported by the terminal). A firmware upgrade is necessary, or the user can force the download by using the **-f** flag.

The error messages "*dmdld: cannot access ...*" or "*dmdld: cannot open ...*" appearing when the named *file* is known to be in the cache, indicate that the application is not available for booting, and *dmdld* cannot find or open the named *file* in the host.

The error message "*dmdld: no memory in terminal*" indicates that the terminal has run out of memory to accept the download. The user may free up memory (by deleting windows, etc..) and re-try.

Other error messages are self-explanatory.

BUGS

The *application-arguments* are not sent to the terminal to update *argc* and *argv* if the named *file* is found in the terminal's application cache.

NAME

dmdman – print manual pages

SYNOPSIS

dmdman [options] [sections] titles

DESCRIPTION

The *dmdman* utility prints the on-line manual pages for the given titles. **Titles** are entered in lower case. The **sections** are numbers from one to eight and correspond to the manual page section numbers. The **section** number may not have a letter suffix. If no **section** is specified, the whole manual is searched for **title** and all occurrences of it are printed.

Options and their meanings are:

-c Preprocess output with col(1).

-d Search the current directory rather than *$DMD/man*.

-w Prints on the standard output only the **pathname** of the entries, relative to *$DMD/man*, or the current directory for *-d* option.

-Tterm

Set *TERM* (refer greek(1)) before printing the manual page. In addition, a null **term** will clear the **term** variable for the duration of the *dmdman* invocation.

-12 Indicates that the terminal type specified on the *-T* option or in the *$Term* variable is to be placed in 12 pitch mode. If this option is used, then the terminal type must not include the "-12" string; otherwise, this string will occur twice in the *$Term* variable, thus making it invalid.

Since the manual pages are not available on the AT&T 3B2 Computer, this command will not work on that computer.

FILES

$DMD/man/?_man/man[1-8]/* Packed manual entries

SEE ALSO

col(1), greek(1) in the *UNIX System V User's Reference Manual.*

NAME

dmdmemory - 630 MTG memory profiler

SYNOPSIS

dmdmemory [-c]

DESCRIPTION

The *dmdmemory* utility presents a graphical representation of the memory usage in the 630 MTG terminal. Memory in the 630 MTG terminal is divided into two types of memory which can be requested by user level function calls:

. non-compactibility *alloc* memory [*alloc*(3R)],

. garbage-collectible *gcalloc* memory [*gcalloc*(3R)].

The *alloc* memory pool starts from the low addressed end of the user memory pool and grows upward. The *gcalloc* memory pool begins at the other end of the user memory pool (i.e. at the highest address) and grows downward. The mobile boundaries of the *alloc* and *gcalloc* pools are named respectively **alloclevel** and **gclevel**.

To contain fragmentation due to the non-compactibility of the *alloc* memory, there is an **alloclimit** which restricts the expansion of the *alloc* pool.

In order to make better use of *alloc* fragments, the 630 MTG memory allocation scheme converts a *gcalloc* request that cannot fit into the *gcalloc* pool to an *alloc* request. If there is a fragment big enough to satisfy the request, it becomes a *gcastray* block. *Gcastray* memory is of the same type as *gcalloc* but resides inside the *alloc* pool. Note that this type of memory is only known by the terminal memory allocation system and cannot be requested directly by a user level function call.

A user is able to monitor all three types of memory with a scope (or zoom) facility, modify the **alloclimit**, and look at the memory or stack usage of a particular process.

The -c option causes *dmdmemory* to be cached in the 630 MTG application cache.

The window size used by *dmdmemory* is fixed. If the window size is not correct, a *Core* icon and a "*menu on button 2*" string are displayed. The window must be reshaped if this appears by selecting either the reshape option from the mouse button 2 menu or the reshape function from the mouse button 3 menu. If reshape is selected from the button 2 menu, the window is automatically reshaped to the proper size. If the reshape function from the button 3 menu is selected, the default window size presented is the correct window for *dmdmemory*.

If a different window size is swept, the core icon and message string are again displayed. Similarly, if at any time the window is reshaped back to an incorrect size, the core icon and string are again displayed. When the window size is incorrect, *dmdmemory* is largely inactive, so at times it may be desirable to place *dmdmemory* in this state to free cpu resources for other processes. Reshaping the window back to the correct size reactivates *dmdmemory*.

In the working mode, the *dmdmemory* window contains the following, from top to bottom:

— Three numerical upper fields which are, from left to right:

the scoped number of *alloc* blocks.
the scoped number of *gcastray* blocks,
the scoped number of *gcalloc* blocks.

The number of scoped blocks is the number of those blocks that fall within the range of memory represented by the dmdmemory window.

—A bar graph which represents the user memory pool in the viewing scope. *Alloc* blocks are reverse-videoed, *gcastray* blocks are *background* textured, and *gcalloc* blocks are grey shaded. To help the visualization, the bar graph is marked by 100 tick marks, 8 pixels apart. The *alloclimit* is represented by a longer vertical bar.

— Five numerical lower fields which are, from left to right:

the starting address of the viewing scope,
the **alloclevel** address,
the **alloclimit** address,
the **gclevel** address,
the ending address of the viewing scope.

In *full view* the viewing scope addresses are the same as the boundary addresses of the total user memory pool. In a *scoped view*, the **alloclevel**, **alloclimit** or **gclevel** addresses may not be displayed if they are out of the viewing scope.

—An alphanumeric field which displays information or help messages. Information includes the scope setting (*full view* or *scoped view*) and the number of bytes per pixel. Help messages depend on the command selected.

The *dmdmemory* facility supports five commands, all accessed from mouse button 2. Note that when a command is being executed, the *dmdmemory* window cannot be reshaped. The five commands are described below.

Base

This command toggles between decimal and hexadecimal bases. Hexadecimal numbers are preceded by a "0x" prefix.

Process

This command changes the mouse cursor to a "target" cursor and asks the user to pick a window by clicking button 1 over it. Picking nothing (i.e., the screen background) or clicking other buttons will cancel the command.

Clicking button 1 over a window will cause the three numerical upper fields to blink. They now represent scoped amounts of *alloc*, *gcastray* and *gcalloc* memory used by the process running inside the selected window. The display inside the bar graph also blinks to mark the corresponding positions of these memory blocks. The alphanumeric field displays the address of the process.

When the *dmdmemory* window is current clicking or holding any button stops the blinking, exits this command mode, and returns *dmdmemory* to the normal viewing mode.

If a cached process is selected, dmdmemory displays only that memory used by the process that was allocated via an alloc or gcalloc procedure call. The memory actually occupied by the process code and data is not displayed by the process command.

Stack

This command is similar to the *process* command, except that the information now deals with the stack assigned to the specified process.

Since the process stack is *alloc*'ed, only the *alloc* field can have a non-zero value (assuming the stack is inside the viewing scope). The alphanumeric field shows how many bytes of its stack a process has used.

The stack command displays the amount of assigned stack space whether the process is cached or not.

Scope

This command allows the user to zoom in (or out) in order to have a closer look at a particular region of the user memory pool.

When this command is selected, a full-length vertical bar blinks at the left side of the bar graph. The user drags the vertical bar by holding down button 1. Note that the starting address of the viewing scope (left-most lower numerical field) also blinks, and changes with new values corresponding to the dragged bar positions. Releasing button 1 will select the new starting address of the viewing scope.

The ending address of the viewing scope is modified in the same manner with a blinking bar appearing at the right end of the bar graph. If this bar is positioned to the left of the previous one, *dmdmemory* takes it as a zoom-out, back to *full view* setting. Otherwise, the action results in a zoomed-in (or scoped) view of the memory pool, and the numerical fields are updated accordingly. Note that some addresses in the lower fields are now out of scope and therefore are not displayed.

During this process, if any button other than button 1 is pressed and held, the command is aborted and the current viewing scope is retained.

The smallest scope is one byte per pixel. Attempts to zoom in further will automatically re-expand the viewing scope to this minimum setting.

Limit

This command allows the user to modify the value of **alloclimit**.

When this command is selected, the vertical bar which represents the position of **alloclimit** in the graphical bar starts to blink. The same thing happens to the **alloclimit** numerical field (middle lower field). To modify its value, the user holds down button 1 and drags the vertical bar to new positions. The numerical field changes accordingly. Note that **alloclimit** cannot go lower than **alloclevel**; otherwise, some *alloc* blocks would be out of the *alloc* pool.

This command does nothing if the **alloclimit** is not within the scope.

Caution: a value for **alloclimit** that is too low restricts the expansion of the *alloc* pool, causing *alloc* requests to fail.

FILES

$DMD/lib/dmdmemory.m downloadable file

SEE ALSO

ucache(1), alloc(3R), gcalloc(3R).

NAME

dmdpi – 630 MTG process inspector and debugger

SYNOPSIS

dmdpi [-c]

DESCRIPTION

Dmdpi is a C language debugger that is bound dynamically to multiple subject processes executing in a 630 MTG window in the layers environment. In order to use *dmdpi* to its full capabilities, it is necessary to compile the source program with the *-g* option of *dmdcc*. However, if the target program is not compiled with *-g*, or no symbol tables at all are available, *dmdpi* works as well as possible with the information provided to it.

If the -c option is selected, *dmdpi* will be cached in the 630 MTG cache system. This will enable *dmdpi* to be executed again without the need for another download.

Dmdpi uses a multi-window user interface. There are three types of windows: debugger control windows, which access the global state of the debugger; process control windows (exactly one per process), which start and stop processes and connect to process-specific functions; and process inspection windows, which include viewers for source text and memory, formatted various ways. Initially, there are three debugger control windows available: **dmdpi**, **help** and **pwd/cd**.

One might need to debug some initialization code that would ordinarily be executed before *dmdpi* has a chance to gain control of the process. The *-z* of *dmdld* is useful for this purpose. This allows you to take control of a process before the first statement is executed. See the *dmdld(1)* manual page for details on using this option.

User Interface

Button 1 points. Pointing at a window makes it current, noted with a highlighted border; pointing at a line of text makes it current and inverts its video. A scroll bar at the left of each window shows how much of the text of a window is visible; pointing into the scroll region and moving the mouse controls what text is displayed.

Button 2 has a menu of operations that apply to the current line. Operations above the ~~~~~ separator are specific to each line; operations below the separator are generic line operations:

cut	Remove the line.
sever	Remove the line and all lines above it.
fold	Wrap the line, if it extends past the right margin.
truncate	Truncate the line at the right margin.

Button 3 has a menu of window-level operations and is in two parts. Operations above the separator are specific to each window. Operations below the separator are the following generic window operations:

reshape Change the size of a window.

move Move a window to a different place.

close Delete a window.

fold Like button 2 fold above except it applies to all lines in the window.

truncate Like button 2 truncate above except it applies to all lines in the window.

top The sub-menu off the **top** is a list of windows; selecting one makes it top and current.

Button 3 is also used to sweep out new windows.

Keyboard characters accumulate at the bottom of the window. If the current line accepts input, it flashes with each keystroke; otherwise, if the current window accepts input, its border flashes. Carriage return is ignored until a line or window accepts the text, whereupon the input line is sent to the line or window.

The following keyboard commands are also available:

'>file' This saves the contents of the current line, or current window if there is no current line, into the named file. To achieve the status of no current line in the window, scroll off the top or bottom of the window.

'<file' Each line of the named file is sent to the line or window as though it had come from the keyboard.

? Each line or window that accepts keyboard input produces some help in response to **?**. These messages specify the format of what may be typed. Items in brackets ([]) are optional parameters in the keyboard input expression. Explanations are contained within braces ({}).

Special cursor icons occasionally appear:

arrow-dot-dot-dot
The host is completing an operation; the terminal is ready asynchronously.

coffee cup
The terminal is receiving input from the host; the terminal momentarily is blocked.

exclamation mark
Confirm a dangerous menu selection by pressing that menu's button again.

Debugger Control Windows

Dmdpi Window

The most important debugger control window is the *dmdpi* window, which is the first window created after the debugger downloads.

Each line within the *dmdpi* window refers to a specific process running in the terminal. A process is identified by its 630 MTG process table address. Along with each process is a path to its host resident download module (argv[0]). This pathname is used by *dmdpi* to find the symbol table and debugger information for the process.

Lines may be introduced to the *dmdpi* window by running *list processes* from the button 3 menu or by typing a process table address and symbol table path from the keyboard. Typing the information at the keyboard may be useful if one wishes to change the pathname of where the process symbol table might be found.

Lines are also introduced into the *dmdpi* window when opening or closing a process control window for a process which is currently not listed in the *dmdpi* window.

Note that the list of processes in the *dmdpi* window is not dynamically updated as new processes are created or deleted, or when application programs exit. This can lead to invalid processes being listed in the *dmdpi* window until list processes is again chosen from the button 3 menu.

Pwd/cd Window

The pwd/cd window controls the working directory of the debugger. The initial working directory is the directory in which *dmdpi* is executed. The working directory can be changed either by typing a path from the keyboard or by selecting directory listings in button 2 and button 3 menus.

Help Window

The help window contains a reminder of user interface mechanics.

Process Control Windows

A process control window is created from the *dmdpi* window in one of two ways. *Open process* on the button 2 menu opens the process currently highlighted in the *dmdpi* window. *Pick process* on the button 3 menu causes the mouse cursor to change to a target cursor which can be used to point to a window containing a process to debug.

The paths to symbol tables shown in the *dmdpi* window are not full path names if the location of the host resident download module for the program being debugged is specified to *dmdld* as a relative path name. If this is the case, the debugger is not able to find symbol tables unless the working directory of the debugger is the same as the directory in which the application was downloaded. If symbol tables cannot be found, close the process window, change the working directory of the debugger from the pwd/cd window, and then reopen the process window.

A process control window indicates the process's state and shows the call stack traceback if the process is halted or dead. The call stack is the

dynamic chain of activation records. The process control window also connects to process inspection windows that access source text, local variables within a stack frame, raw memory, and so on. These windows are cross-connected; so, for example, an instruction in a process's assembly language window can be inspected as a hexadecimal opcode in the raw memory window. Closing the process control window closes all the process inspection windows associated with it.

States are:

RUNNING	running normally
STOPPED	stopped asynchronously by the debugger
BREAKPOINT	halted on reaching breakpoint
STMT STEPPED	halted after executing C source statement(s)
INSTR STEPPED	halted after executing machine instruction(s)
PROCEESS EXCEPTION	a process exception has occurred
ERROR STATE	the process has probably exited.

When in the **RUNNING** state, the status of selected bits of the P->state variable is displayed and updated.

The menu operations on the process are:

run	let the process run
stop	stop the process
src text	open source text window(s)
Globals	open window for evaluating expressions in global scope
RawMemory	open window for editing uninterpreted memory
Assembler	open window for disassembler
User Types	open window for setting user types
Journal	open debugging session journal window
Bpt List	open breakpoint list window

Each line of the call stack traceback describes one function. Each function in the traceback can open a stack frame expression evaluator window or display its current source line.

Process Inspection Windows

Source Text Windows

The source text window contains a listing of a source file. If there is more than one source file for the process, selecting the **src text** item in the process control window will give you a source files window in which there is a listing of all the source files associated with that process. Library function source files are included in this list, even though one might not actually have the source for these functions. By highlighting a source line and selecting **open source file** in the button 2 menu, you can open a source listing for that file. Each source file is in a separate window, which can be opened when needed. The source files are searched for in the working directory. Entering a pathname from the keyboard (when the Source files window is current) enters a pathname prefix which points to a directory where the source can be found, without changing the working directory.

When opening a source file, *dmdpi* checks to see whether the source file is in time sync with the object module. If not, *dmdpi* gives a message of this fact. One may override this condition with the **reopen** item in the button 3 menu of the source text window. Source lines are displayed on a "per request basis." In other words, only the lines that are currently visible are sent from the host. More lines are sent to be displayed on the terminal as needed.

Specific strings may be searched for in the source text by using */string*, or the *?string* entered at the keyboard, for searching forward and backward in the source text respectively. The search will begin at the next (previous for backwards search) C language statement rather than at the next source line. Note that repeated reverse searches for the same pattern must be specified as *??* rather than *?* due to a conflict with the help operator (?). Line numbers can also be searched for by entering a line number at the keyboard when a line is not current within the window. If a line is current, the number is evaluated as a constant expression (see expressions below). To achieve the status of no current line, scroll the current line off the top or bottom of the window.

Breakpoints are set on source lines. A breakpoint is set by highlighting the line on which you wish to break execution and selecting **set bpt** from the button 2 menu. A breakpoint is denoted by a '>>>' next to the source line. When the process reaches this line the process halts and will not execute the line on which the breakpoint is set. Clearing the breakpoint is done by highlighting the line on which a breakpoint is set and selecting **clear bpt** from the button 2 menu. Clearing the breakpoint can also be done from the breakpoint list window (see below). A **conditional breakpoint** is a breakpoint that is set with a certain condition. When this condition evaluates to TRUE, the process is halted. Any valid *dmdpi* expression may be used as a condition (see expressions). To set a conditional breakpoint, select **cond bpt** from the button 2 menu. You are prompted to enter an expression from the keyboard as a condition. An example of a condition would be (x==1). When the variable x becomes equal to 1, then execution breaks. The **trace on** item in the button 2 menu is actually a conditional breakpoint with the condition of 0, meaning that the condition never evaluates to TRUE. This has the effect of tracing a statement but never breaking execution. The conditional breakpoint is removed in the same way a regular breakpoint is removed.

Once the process has been halted, select **run** to start the process running again. You can also **step** (execute) a number of source lines and then stop again after these statements have been executed. When statements are stepped, the debugger will not enter functions unless the **step into fcn** item is actually specified. The current statement can always be seen by selecting the **current stmt** item in the button 3 menu. This highlights the statement currently in the PC.

Another option that is available in the source text window is the ability to look at the assembly code for a specified line. Highlighting a line and selecting **assembler** in the button 2 menu displays the first assembler instruction of the statement.

Globals and Stack Frame Windows

A stack frame window is opened from a line in the call stack traceback in the process control window or from a line of source text. A globals window is opened from the button 3 menu in the process control window. These windows evaluate expressions with respect to global scope, and scope in a function, respectively.

Expressions

Expressions may be entered from the keyboard or with the mouse. The syntax for expressions in *dmdpi* is the same as C language expressions, except for differences noted below. The expressions are most commonly used for inspecting values of variables in the program that is being debugged. An example of an expression is *r.origin.x*. This may be typed in order to inspect the x coordinate value of a rectangle origin point if the process has a rectangle *r*.

A summary of *dmdpi*'s expression syntax is presented here only to aid comprehension, rather than an exact statement of the language.

```
expression :
        constant
        primary
        *expression
        &expression
        -expression
        !expression
        ~expression
        sizeof expression
        typeof expression
        fabs (expression)
        (type-name) expression    /* from menu only */
        {expression} identifier
        expression binop expression
        expression = expression
        expression , expression


primary:
        $
        identifier
        ( expression )
        primary ( [expression-list] )
        primary[ expression ]
        lvalue.identifier
        primary -> identifier
```

lvalue:

identifier
primary[expression]
lvalue.identifier
primary -> identifier
**expression*
(lvalue)

binop:

* / % + - >> << < > <= >= == != & ^
| && ||

The major differences in the expressions which *dmdpi* understands and the C expressions are:

The unary operators *fabs* and *typeof* are supported. *fabs* evaluates the absolute value of a floating point number. *typeof* evaluates the type of an expression. Examples are:

fabs(-2.0)=2
typeof(r.origin)= struct Point

The concept of a "current expression" has been introduced with the **$** operator. **$** is equal to the current highlighted expression. For example, if the line containing *r.origin* is highlighted, one may type **$.x** to see the value of the x coordinate. Another example of the **$** expression is **$=<expression>**. This can be used, for instance if **$** is equal to a variable x and you wish to change the value of x to <expression>.

Expressions are evaluated within the scope boundaries of the window in which they are typed. One can cross scope boundaries in order to evaluate an expression with the syntax { expression } function-name. This, for example, is useful for using the globals window to look at static variables that are local to a function without having to open up a stack frame window.

Type casting may only be done through the use of the menu.

The following is not supported by *dmdpi*: ++ -- ?: op= string.

NOTE: expressions are always evaluated internally with a 32-bit precision. Therefore, results may not correspond in all cases with those generated by a C program.

Expressions are also used to specify the condition in the conditional breakpoint. Note that the C comma operator is very useful in specifying the condition. Expressions separated by a comma are evaluated left-to-right and all but the rightmost expression are discarded. For example, a condition of *(x,y,x==y)* evaluates all three expressions; however, only the last expression (x==y) determines

the result of the overall condition. The result is that the values of x and y are printed but execution halts only when x==y.

Registers in the stack frame windows are prefixed with the character **$** , for example, **$d0.** The address of a register is the location at which it was saved. Register values are only available after execution has been halted at a breakpoint or after a step. The exception to this rule is that one may look at register variables in calling functions if they happened to be saved in the called function.

An expression may be made *spy*, in order to observe changes in the expression. The value of a spy expression is evaluated and displayed each time the debugger looks at the process, i.e., when the process calls wait() or sleep(). If the value of a spy changes, the expression is updated and a message is given that the expression has changed. If a conditional breakpoint (or trace on) is set, then the process will be halted. The option **changed spies** in the button 2 menu will manually force all spies to be re-evaluated.

A maximum of 150 global variables will fit into the globals menu. If the targeted program has more than 150 global variables, the remaining variables must be accessed by typing their name from the keyboard.

Raw Memory Window

The raw memory window is a "memory editor" in which memory is viewed as a sequence of 1-, 2-, 4- or 8-byte cells. The **'.'** operator is a special symbol which denotes a cell address. Therefore, commands such as **.+1** in the button 2 menu give the next increment of memory after the current cell address. The keyboard command **.=<expression>** displays the cell with address equal to expression. The expression syntax is the same as defined above. The format of the displayed memory cells is *x/y: <contents>*, where x is the cell address, and y is the viewing increment.
Some of the functions available are:

change cell size and display format
: Use the **size** and **format** items in the menu.

display cells above and below current cell.
: Use the **.[+-]<amount>** options in the menu.

indirect to cell
: Look at the cell using the contents of the current cell as an address. Use the *** thru .** option.

change cell value
: This is done with the keyboard expression: **$=<expression>**

spy on memory cell
: If the memory contents change, *dmdpi* will give notification.

disassemble instruction at cell.

Display the assembler instruction in the assembler window. Use the **asmblr** option in the button 2 menu.

(Dis)Assembler Window

Allows viewing of memory as a sequence of assembler instructions. The menu options of this window are similar to those in the source text window. The difference is that this window deals with assembler instructions rather than the high-level source code.

An instruction at a certain address can be displayed by entering the keyboard expression .=<**expression**>. The expression syntax is the same as defined above. More instructions can be viewed in a sequential manner using the **next** options in the button 2 menu. The next 1, 5, or 10 instructions starting from the current instruction can be displayed.

When setting a breakpoint or stepping into an assembler function, one must step through the link and the movm.l instructions before *dmdpi* will be able to generate the stack frame for the function.

Some of the other functions available are:

change display format

open a stack frame window for instruction's function

display instruction as cells in the raw memory window

set/clear breakpoint on instruction

display instruction at current PC

single step instruction(s)

User Types Window

Shows user-defined types and allows the display format of user-defined types displayed in the globals and stack frame windows to be changed. For example, the display format of a structure may be changed so that certain fields are not displayed (hidden) and other fields are displayed (shown).

Journal Window

Keeps a log of significant events in the course of a debugging session.

Breakpoint List Window

Lists all active source and assembler breakpoints. Allows clearing of specified breakpoints or all breakpoints.

Functions available include:

show source or assembler line at which a breakpoint is set

clear a single breakpoint

clear all breakpoints

SEE ALSO

dmdcc(1), dmdld(1), ucache(1).

WARNINGS

Do not use the -O optimizer option of dmdcc when compiling a program to be debugged with *dmdpi*. This will confuse *dmdpi*.

It is possible to receive a message that there is no more memory on the host system. This will happen if the process you are debugging has a very large symbol table, or if you are debugging many processes at the same time. The maximum amount of memory that a UNIX process is allowed to consume can be changed by a system administrator. For a 3B2 host computer running System V Release 2.0, how to change the per process memory limit is documented in the manual *AT&T 3B2 Computer Unix System V Release 2.0 System Administration Utilities Guide* in the chapter "Administrative Tasks" under "Tunable Parameters." An alternative to changing the host computer's per process memory limit is to use the *mc68cprs* CCS utility to compress the size of process symbol tables before they are opened for debugging with *dmdpi*.

BUGS

In switch statements there is no boundary between the last case and the branch code; the program *appears* to jump to the last case (but is really in the branch) and then to the real case.

The structure P which is of type "struct Proc *" within applications is interpreted by *dmdpi* as "struct proc". This implies that one must type P.state rather than P->state when accessing the structure P from the keyboard.

If a program contains multiple global structure declarations of the same name, *dmdpi* will ignore all but the first declaration.

A breakpoint cannot be set on a goto or return statement. Attempting to do so will set a breakpoint on the following line. Also, stepping onto a goto or return statement will execute the goto or return instead of stopping on the line.

When stepping past an if statement that is the last statement within a while loop and the if condition fails and does not have an else condition, the program will appear to jump to the last line within the if statement. It is really jumping to the statement that will branch back to the top of the while loop.

NAME

dmdversion – inquire terminal/host software version

SYNOPSIS

dmdversion [**-ehlst**]

DESCRIPTION

The *dmdversion* utility displays the version numbers of the 630 MTG terminal and host software. The terminal version number is the equivalent to the ASCII string which contains three fields (f1;f2;f3) defined as follows:

f1	identifies the 630 MTG as a windowing terminal
f2	identifies the terminal as a 630 MTG
f3	identifies the firmware release

Host software version is read from the file $DMD/VERSION.

The **-t** option is used to display the terminal version number. The **-h** option is used to display the host software version number. The default action is to display both terminal and host software version numbers.

In the **layers** environment, terminal version is found through an *ioctl(2)* call to the xt device driver. In non-layers, or if the **-e** flag is specified, the terminal version is found through the Request Terminal Type escape sequence **ESC[c**.

The **-l** option can be used to inquire if Local Area Network (LAN) Encoding is set for the terminal through terminal setup. This is found through the Request Encoding escape sequence **ESC[F**. This option excludes the **-e** option and does not inquire the terminal version number.

When the **-s** flag is present, no output is printed but an exit value is returned as follows. If the **-t** or **-e** options are present, the decimal ascii value of the last digit of the terminal's version is returned. If the **-l** option is present, 1 is returned if LAN encoding is enabled; 0 otherwise. The **-l** option will overide the **-t** or **-e** options. In all other cases, 255 (-1) is returned.

EXAMPLE

The following example can be used to determine if a 630 MTG or some other windowing terminal (such as a 5620) is being used.

```
case `dmdversion -t` in
    *'8;8'*)
          echo I am a 630
          ;;
    *'8;7'*)
          echo I am a 5620
          ;;
    *)
          echo Unknown terminal type
          ;;
esac
```

FILES

$DMD/VERSION the host version

SEE ALSO

version(3R).
ioctl(2) in the *UNIX System V Programmer's Reference Manual.*
layers(1) in the UNIX System V Release 3 User's Reference Manual.
layers(1) in the 5620 Dot-Mapped Display Reference Manual.
630 MTG Terminal User's Guide.

DIAGNOSTICS

The **-e** and **-l** flags only work if the window connected to the standard output is running the default 630 MTG terminal emulator or any other emulator that supports the described escape sequences.

NAME

icon – interactive icon drawing program

SYNOPSIS

icon [**-x** m] [**-y** n] [**-c**]

DESCRIPTION

The *icon* utility is a menu-driven interactive icon and picture drawing program. It runs under *layers* using the "mouse" and keyboard for command and text entry. The default *icon* display consists of a 50X50 cell grid in the lower right-hand corner of the layer in which *icon* is invoked. By invoking *icon* with the **-x** and **-y** flags, the grid size may be specified to be *mXn*, overriding the default. Each cell in this grid corresponds to a single bit in the *icon* being created or edited. In the upper left-hand corner of the layer, an actual size view of the icon is displayed.

The **-c** option causes icon to be cached in the 630 MTG application cache.

The grid size parameters *m* and *n* must be in the range of 1 to 480. A parameter larger than 480 is reduced to 480, and a parameter smaller than 1 is set to the default value of 50.

If icon is invoked in a window that is too small for the specified grid size, icon will display a grid icon in the upper left corner of the window along with a message "menu on button 2". The window must be reshaped before icon will continue. At this point a menu on button 2 will contain a reshape selection. If selected, icon will automatically reshape its window to a size and shape appropriate for the grid size. Alternately, the button 3 reshape function can be used to manually reshape the window. If at any time the window is again reshaped to a size too small to display the selected grid size, icon will redisplay the grid icon and the "menu on button 2" message.

When using icon, the meanings of the three mouse buttons are as follows:

Button 1 Button 1, when depressed and held in, fills in the grid position pointed to by the mouse cursor.

Button 2 Button 2, when depressed and held in, clears the grid position pointed to by the mouse cursor.

Button 3 Button 3, when depressed, displays a matrix of icons. By moving the cursor (now a box) over the desired icon and releasing the button, commands will be invoked. If the command requires a section of the grid display to be selected, depressing button 2 will select a 16x16 grid outline. To specify other than this 16x16 grid outline, depress button 3 and sweep out the rectangle you wish the command to act on.

Commands

The command selection matrix icons are described below from the upper left by rows to the bottom right. On the bottom row is a help command designated by the word "help."

Arrow	move selection to another portion of the grid.
Copier	copy selection to another portion of the grid.
Black and white squares	change light squares to dark and dark squares to light (invert video).
Garbage can	erase.
Horizontal wrap arrow	flip on the x-axis.
Vertical wrap arrow	flip on the y-axis.
To right and down arrow	rotate 90 degrees clockwise.
Up and to left arrow	rotate 90 degrees counterclockwise.
Horizontal lines	shear along the x-axis.
Vertical lines	shear along the y-axis.
Four line sets	stretch (expand).
Pinwheels	take one pattern and make many copies of it.
Eyeglasses	read an icon file.
Grid	draw a reference grid.
Mouse	change current mouse cursor to selected 16 x 16 grid.
Quill pen	write an icon file.
Grid, arrow to grid	bitblt operator.
HELP	prints the help menu.
Smoking gun	exit the *icon* program.

Cursor Icons

The following are status indicator icons that the mouse cursor changes to under various conditions:

Alarm clock	wait.
Dead Mouse	mouse inactive.
Dark square in stack	menu on button 3.
Square with arrow	sweep rectangle (button 3).
Double square with arrow	sweep rectangle (button 3) or get 16x16 grid frame (button 2).

FILES

$DMD/lib/icon.m	terminal support program
$DMD/icons/*	icons

SEE ALSO

ucache(1).
layers(1) in the *UNIX System V Release 3 User's Reference Manual.*
layers(1) in the *5620 Dot-Mapped Display Reference Manual.*

NAME

jim, jim.*– 630 MTG text editor

SYNOPSIS

jim [**-c**] [files . . .]
jim.* [**-f**] [**-t**] [files . . .]

DESCRIPTION

Jim is the text editor for the 630 MTG terminal.

It is a shared cached application if the **-c** option is specified. This means that multiple instances of jim may run simultaneously without needing to do a download for each instance. Once jim is downloaded it does not have to be downloaded again.

Jim relies on the mouse to select text and commands; it runs only under *layers*(1). *Jim*'s screen consists of a number of *frames*, a one-line command and diagnostic frame at the bottom, and zero or more larger file frames above it. Except where indicated, these frames behave identically. One of the frames is always the current frame, to which typing and editing commands refer, and one of the file frames is the working frame, to which file commands such as pattern searching and I/O refer.

A frame has at any time a selected region of text, indicated by reverse video highlighting. The selected region may be a null string between two characters, indicated by a narrow vertical bar between the characters. The editor has a single *save buffer* containing an arbitrary string. The editing commands simply invoke transformations between the selected region and the save buffer.

The mouse buttons are used for the most common operations. Button 1 is used for selection. Clicking button 1 in a frame which is not the current frame makes the indicated frame current. Clicking button 1 in the current frame selects the null string closest to the mouse cursor. Making the same null selection twice ('double clicking') selects (in decreasing precedence) the bracketed or quoted string, word, or line enclosing the selection. By depressing and holding button 1, an arbitrary contiguous visible string may be selected. Button 2 provides a small menu of text manipulation functions, described below. Button 3 provides control for inter-frame operations.

The button 2 menu entries are:

cut Copy the selected text to the save buffer and delete it from the frame. If the selected text is null, the save buffer is unaffected.

paste Replace the selected text by the contents of the save buffer.

snarf Copy the selected text to the save buffer. If the selected text is null, the save buffer is unaffected.

Typing replaces the selected text with the typed text. If the selected text is not null, the first character typed forces an implicit **cut**. Control characters are discarded, but BS (control H), ETB (control W), NL (control J) and ESC (escape) have special meanings. BS is the usual backspace character, which erases the character before the selected text (which is a null string when it takes effect). ETB erases back to the word boundary preceding the selected text. There is no line kill character. NL toggles the current frame between the workframe and the diagnostic frame, and can be a substitute for manual frame selection with the mouse. ESC selects the text typed since the last button hit or ESC. If an ESC is typed immediately after a button hit or ESC, it is identical to a **cut**. ESC followed by **paste** provides the functionality of a simple undo feature.

The button 3 menu entries are:

new Create a new frame by sweeping with the mouse.

reshape
Change the shape of the indicated frame. The frame is selected by clicking button 3 over the frame.

close Close the indicated frame and its associated file. The file is still available for editing later; only the associated frame is shut down.

write Write the indicated frame's contents to its associated file.

The rest of the menu is a list of file names available for editing. To work in a different file, select the file from the menu. If the file is not open on the screen, the cursor will switch to an outline box to prompt for a rectangle to be swept out with button 3. (Clicking button 3 without moving the mouse creates the largest possible rectangle.) If the file is already open, it will simply be made the workframe and current frame (for typing), perhaps after redrawing if it is obscured by another frame. The format of the lines in the menu is:

- possibly an apostrophe, indicating that the file has been modified since last written,
- possibly a period or asterisk, indicating the file is open (asterisk) or the workframe (period),
- a blank,
- and the file name. The file name may be abbreviated by compacting path components to keep the menu manageable, but the last component will always be complete.

The work frame has a *scroll bar* — a black vertical bar down the left edge. A small tick in the bar indicates the relative position of the frame within the file. Pointing to the scroll bar and clicking a button controls scrolling operations in the file:

Button 1 Move the line at the top of the screen to the y position of the mouse.

Button 2 Move to the absolute position in the file indicated by the y position of the mouse.

Button 3 Move the line at the y position of the mouse to the top of the screen.

The bottom line frame is used for a few typed commands, modeled on *ed*(1), which operate on the workframe. When a carriage return is typed in the bottom line, the line is interpreted as a command. The bottom line scrolls, but only when the first character of the next line is typed. Thus, typically, after some message appears in the bottom line, a command need only be typed; the contents of the line will be automatically cleared when the first character of the command is typed. The commands available are:

e *file* Edit the named *file*, or use the current file name if none specified. Note that each file frame has an associated file name.

E *file* Edit the named *file* unconditionally, as in *ed*(1).

f *file* Set the name of the file associated with the work frame, if one is specified, and display the result.

g *files* Enter the named *files* into the filename menu, without duplication, and set the work frame to one of the named files. If the new work frame's file is not open, the user is prompted to create its frame. The arguments are passed through *echo*(1) for shell metacharacter interpretation.

w *file* Write the named *file*, or use the current file name if none specified.

q Quit the editor.

Q Quit the editor unconditionally, as in *ed*(1).

/ Search forward for the string matching the regular expression after the slash. If found, the matching text is selected. The regular expressions are exactly as in *egrep*(1), with two additions: the character '@' matches any character *including* newline, and the sequence '\n' specifies a newline, even in character classes. The negation of a character class does not match a newline. An empty regular expression (slash-newline) repeats the last regular expression.

? Search backwards for the expression after the query.

94 Select the text of line 94, as in *ed*.

$ Select the text of the last line.

cd *dir* Set the working directory to *dir*, as in the shell. There is no CDPATH search, but $HOME is the default *dir*.

= Display the line number of selection in the current frame.

>Unix-command
Sends the selected text to the standard input of *Unix-command*.

<Unix-command
Replaces the selected text by the standard output of *Unix-command*.

|Unix-command
Replaces the selected text by the standard output of *Unix-command*, given the original selected text as standard input.

If any of <, > or | is preceded by an asterisk *, the command is applied to the entire file, instead of just the selected text. If the command for < or | exits with non-zero status, the original text is not deleted; otherwise, the new text is selected. Finally, the standard error output of the command, which is merged with the standard output for >, is saved in the file $HOME/jim.err . If the file is non-empty when the command completes, the first line is displayed in the diagnostic frame. Therefore the command ">pwd" will report *jim*'s current directory.

The most recent search command ('/' or '?') and Unix command ('<', '|', or '>') are added to the button 2 menu, so that they may be easily repeated.

Attempts to quit with modified files, or edit a new file in a modified frame, are rejected. A second 'q' or 'e' command will succeed. The 'Q' or 'E' commands ignore modifications and work immediately. Some consistency checks are performed for the 'w' command. *jim* will reject write requests which it considers dangerous (such as writes that change files which are modified when read into memory). A second 'w' will always write the file.

If *jim* receives a hang-up signal, it writes a recover file, which is a shell command file that, when executed, will retrieve files that were being edited and had been modified. The name of the file will be of the form *jim.* followed by a uniquely generated alphanumeric string. *Jim* will send mail to the logon id saying files may be recovered and specifying the path and name of the recover file. If it cannot write this file in the home directory, it writes it in the current working directory. The **-t** option prints a table of contents. By default, the jim recover file is interactive; the **-f** option suppresses the interaction. If no *file* argument is given to the jim.recover shell file, the recovery will apply to all modified files at the time when jim received the hang-up signal. If there is a *file* argument, only those files will be recovered.

FILES

$DMD/lib/jim.m	terminal support program
/tmp/jim.*	temporary file
$HOME/jim.err	diagnostic output from Unix commands
jim.*	recovery script created upon *jim* failure

SEE ALSO

ucache(1).
ed(1), echo(1), egrep(1) in the *UNIX System V User's Reference Manual.*
layers(1) in the *Unix System V Release 3 User's Reference Manual.*
layers(1) in the *5620 Dot-Mapped Display Reference Manual.*

WARNING

Jim is reshapable, but a reshape clears the screen space of all open frames.

BUGS

The regular expression matcher is non-deterministic (unlike *egrep*), and may be slow for complicated expressions.

The < and ¦ operators don't snarf the original text.

NAME

jx – 630 MTG execution and stdio interpreter

SYNOPSIS

jx [**-d**] [**-p**] [**-z**] [**-f**] [**-Z** n] [**-n**] file
[command line arguments]

DESCRIPTION

The *jx* utility downloads the program in *file* to the 630 MTG and runs it there, simulating most of the standard I/O library functions. This gives application programs downloaded into the 630 MTG the ability to perform operations such as file I/O to files resident on the host computer, using the same interface as programs written for execution on the host computer.

The jx utility calls *dmdld* to do the download into the terminal. Therefore, the *-d, -p, -z, -f, -Z,* and *-n* options are available for use with *jx*. See the *dmdld(1)* manual page for information on these options.

Stdin directed to the host portion of *jx*, either through the *jx* command line or with the *popen* function, is properly redirected. Note that input from the 630 MTG keyboard is not translated to *stdin* to the host portion of *jx*. Rather, programs wishing to read from the keyboard should use *kbdchar*(3R).

Stdout and *stderr*, written to by the below library functions, will be stored in a buffer during execution. After the terminal program has been rebooted, *stdout* and *stdin* will be redirected to the terminal.

Programs intended for use by *jx* should include <**dmd.h**> and <**dmdio.h**> and call *exit*(3R) upon termination. *Exit*() returns control to the shell and causes a reboot of the default terminal emulator.

What follows is a list of stdio library functions available under *jx*. These functions are called from an application downloaded into the 630 MTG by *jx*. The *jx* library routines in the terminal then translate the call into a message which is sent to the host portion of *jx* for processing.

getc	getchar	fgets	fflush
putc	putchar	puts	fputs
fopen	freopen	fclose	access
popen	pclose	fread	fwrite
printf	Printf	fprintf	Fprintf

The functions fprintf and printf are stripped down versions of those on UNIX. The functions that start with an upper case letter are identical to those on UNIX. See printf(3L) for more details.

FILES

$DMD/include/dmdio.h	
$DMD/lib/sysint	host portion of *jx* after download is complete
$HOME/.jxout	saved standard output
$HOME/.jxerr	saved standard diagnostic output

SEE ALSO

dmdld(1), exit(3R), kbdchar(3R), printf(3L).
access(2), fopen(3S), fread(3S), getc(3S), popen(3S), printf(3S), putc(3S), puts(3S) in the *UNIX System V Programmer's Reference Manual.*

WARNING

Because 630 MTG keyboard data is not sent to the *stdin* of the host component of *jx*, applications running under *jx* which read from the *stdin* will hang if their *stdin* is not redirected.

The *stdin* can be redirected either from the command line or by function calls inside the application process running on the 630 MTG terminal.

BUGS

Jx does not work when su'ed to another user.

Jx does not work in the nonlayers environment.

Jx does not work with application cached with **A_SHARED**, **A_BSS** or **A_DATA**.

Stderr is buffered. Use fflush (stderr) if this is a problem.

getc(), *getchar()*, *putc()*, and *putchar()* are not macros as described in *getc(3S)* and *putc(3S)*.

The *fopen()* routine does not support the following modes: r+, w+, a+.

NAME

loadfont – font managing program

SYNOPSIS

loadfont [-r name,name...] [-p directory] [-s] [-c] [file...]

DESCRIPTION

The *loadfont* program lets the user load and remove fonts from the terminal's cache. It will download its terminal side which interacts with the user. The command line arguments have the following meanings:

-r Removes the given font from the terminal's cache.

-p Sets default search path for font files. If this isn't set, it defaults to $DMD/termfonts. All the file names in this directory are put in the load submenu.

-s Makes loadfont stay running after executing the earlier command line options.

-c Causes loadfont to be cached in the 630 MTG cache system. When this option is used, the loadfont program will remain in the terminal after the program exits. Then, the next time loadfont is executed, it will not have to be downloaded again.

Another effect of downloading loadfont with the –c option is that it can be executed in more than one window without additional downloading.

file Loads the given font file into the terminal's cache.

If a **file** argument and/or the **-r** option are given, loadfont will exit after loading/removing the specified fonts, unless the **-s** is also given to make it stay running.

The user interacts with *loadfont* by using a button 2 menu and, when prompted, the keyboard. When button 2 is depressed, a menu with the following items appears.

Load Load has a submenu of fonts that can be downloaded. The fonts listed are all the files in the default search directory. If the first item, Keyboard, is selected, the user is prompted for the name of a font file to download.

Remove Remove has a submenu of all the fonts in the terminal's cache. Selecting one will remove it from the cache and free the memory it was using. Fonts in ROM or being used by another process will be greyed in the menu and cannot be removed.

Quit Will make the program exit.

While a font is downloading, the button 2 menu will have only one item, Terminate Download. Selecting it will stop the download and switch the user back to the original menu.

The three fonts in ROM, resident to the 630 MTG, are called "small font," "medium font," and "large font." Brief descriptions of these fonts follow.

In the tables, *cell size* indicates the dimensions of the rectangle containing the character image. All the characters in a particular font have the same cell size. *Character size* is the dimension of the largest character in the font. *Max chars* indicates how many characters in that font size will fit horizontally and vertically in a full screen window. See the section on fonts in the *630 MTG Software Development Guide* for more information on font data structures.

Font Name	small font
Character Size	6 pixels wide by 9 pixels high
Cell Size	7 pixels wide by 14 pixels high
Max Chars	140 across, 69 down
Font Name	medium font
Character Size	8 pixels wide by 11 pixels high
Cell Size	9 pixels wide by 14 pixels high
Max Chars	109 across, 69 down
Font Name	large font
Character Size	9 pixels wide by 12 pixels high
Cell Size	11 pixels wide by 16 pixels high
Max Chars	89 across, 61 down

FILES

The default search path for font files is $DMD/termfonts. This directory contains nine fonts.

Font Name	noseprint
Character Size	5 pixels wide by 7 pixels high
Cell Size	6 pixels wide by 9 pixels high
Max Chars	163 across, 108 down
Comments	The characters are a single pixel thick and all are smaller than the characters found in the resident "small font".
Font Name	7x14thin
Character Size	6 pixels wide by 9 pixels high
Cell Size	7 pixels wide by 14 pixels high
Max Chars	140 across, 69 down
Comments	The difference between "7x14thin" and the resident "small font" is in the thickness of the characters. The "7x14thin" consists of characters which are a single pixel in thickness whereas the "small font" consists of characters which are two pixels in thickness.

Font Name	12x18norm
Character Size	10 pixels wide by 13 pixels high
Cell Size	12 pixels wide by 18 pixels high
Max Chars	81 across, 54 down
Comments	The characters are 2 pixels thick.
Font Name	12x25thin
Character Size	9 pixels wide by 18 pixels high
Cell Size	12 pixels wide by 25 pixels high
Max Chars	81 across, 39 down
Comments	The characters are a single pixel thick.
Font Name	12x25norm
Character Size	9 pixels wide by 18 pixels high
Cell Size	12 pixels wide by 25 pixels high
Max Chars	81 across, 39 down
Comments	The characters are 2 pixels thick.
Font Name	12x25round
Character Size	9 pixels wide by 18 pixels high
Cell Size	12 pixels wide by 25 pixels high
Max Chars	81 across, 39 down
Comments	The main difference between this font and the font called "12x25norm" is in the STYLE of the characters. The characters in "12x25round" are rounder in appearance, whereas the characters in "12x25norm" are squarer in appearance.
Font Name	12x25BOLD
Character Size	9 pixels wide by 18 pixels high
Cell Size	12 pixels wide by 25 pixels high
Max Chars	81 across, 39 down
Comments	The characters are 3 pixels thick.
Font Name	script
Character Size	15 pixels wide (approx.) by 15 pixels high
Cell Size	16 pixels wide by 24 pixels high
Max Chars	61 across, 40 down
Comments	The characters in this font are created in script style.
Font Name	twice_big
Character Size	18 pixels wide by 24 pixels high
Cell Size	22 pixels wide by 32 pixels high
Max Chars	44 across, 30 down
Comments	The characters are twice as tall and twice as wide as the resident "large font"

SEE ALSO

ucache(1), font(4).
630 MTG Software Development Guide.

WARNING

Fonts that do not have 128 characters require the proper parity setting when used in the default window process in the non-layers environment. This is needed because a mod is done on characters received from the host with the number of characters in the font being used.

This implies that parity bits are ignored for fonts with exactly 128 characters, since c&0x7F == c%128. Fonts with less than 128 characters need identical parity settings on the host and in the terminal so that parity bits get stripped properly. Characters with more than 128 characters require 8 bits with no parity on both the host and in the terminal, because the eighth bit is used to access characters greater than 128.

NAME

mc68ar – archive and library maintainer for portable archives

SYNOPSIS

mc68ar key [posname] afile name ...

DESCRIPTION

Mc68ar maintains groups of files combined into a single archive file. Its main use is to create and update library files as used by the link editor. It can be used, though, for any similar purpose.

When *mc68ar* creates an archive, it creates headers in a format that is portable across all machines. The portable archive's format and structure are described in detail in *ar*(4). The archive symbol table [described in *ar*(4)] is used by the link editor [*mc68ld(1)*] to effect multiple passes over libraries of object files in an efficient manner. Whenever the *mc68ar*(1) command is used to create or update the contents of an archive, the symbol table is rebuilt. The symbol table can be forced to be rebuilt by the **s** option described below.

Key is one character from the set **drqtpmx**, optionally concatenated with one or more of **vuaibcls**. *Afile* is the archive file. The *names* are constituent files in the archive file. The meanings of the *key* characters are:

d Deletes the named files from the archive file.

r Replaces the named files in the archive file. If the optional character **u** is used with **r**, then only those files with modified dates later than the archive files are replaced. If an optional positioning character from the set **abi** is used, then the *posname* argument must be present and specifies that new files are to be placed after (**a**) or before (**b** or **i**) *posname*. Otherwise new files are placed at the end.

q Quickly appends the named files to the end of the archive file. Optional positioning characters are invalid. The command does not check whether the added members are already in the archive. Useful only to avoid quadratic behavior when creating a large archive piece-by-piece.

t Prints a table of contents of the archive file. If no names are given, all files in the archive are tabled. If names are given, only those files are tabled.

p Prints the contents of named files in the archive.

m Moves the named files to the end of the archive. If a positioning character is present, then the *posname* argument must be present and, as in **r**, specifies where the files are to be moved.

x Extracts the named files. If no names are given, all files in the archive are extracted. In neither case does **x** alter the archive file.

v Verbose. Under the verbose option, *mc68ar* gives a file-by-file description of the making of a new archive file from the old archive and the constituent files. When used with **t**, it gives a long listing of all information about the files. When used with **x**, it precedes each file with a name.

c Create. Normally, *mc68ar* will create *afile* when it needs to. The create option suppresses the normal message that is produced when *afile* is created.

l Local. Normally, *mc68ar* places its temporary files in the directory **/tmp**. This option causes them to be placed in the local directory.

s Symbol table creation. Forces the regeneration of the archive symbol table even if *mc68ar*(1) is not invoked with a command which will modify the archive contents. This command is useful to restore the archive symbol table after the *mc68strip*(1) command has been used on the archive.

FILES

/tmp/ar* temporaries

SEE ALSO

mc68ld(1), mc68lorder(1), mc68strip(1).
a.out(4), ar(4) in the *UNIX System V Programmer's Reference Manual.*

BUGS

If the same file is mentioned twice in an argument list, it may be put in the archive twice.

NAME

mc68as – MC68000 assembler

SYNOPSIS

mc68as [**-o** objfile] [**-n**] [**-m**] [**-R**] [**-V**] file-name

DESCRIPTION

The *mc68as* command assembles the named file. The following flags may be specified in any order:

-o *objfile* Puts the output of the assembly in *objfile*. By default, the output file name is formed by removing the **.s** suffix, if there is one, from the input file name and appending a **.o** suffix.

-n Turns off long/short address optimization. By default, address optimization takes place.

-m Runs the *m4* macro pre-processor on the input to the assembler.

-R Removes (unlinks) the input file after assembly is completed.

-V Writes the version number of the assembler being run on the standard error output.

FILES

/tmp/mc68a[A-L]AAa*XXXXXX* temporary files

SEE ALSO

mc68ld(1), mc68nm(1), mc68strip(1).
a.out(4) in the *UNIX System V Programmer's Reference Manual.*
m4(1) in the *UNIX System V User's Reference Manual.*
UNIX Assembler User's Guide for the Motorola 68000 in the *630 MTG Software Development Guide.*

WARNING

If the **-m** (*m4* macro pre-processor invocation) option is used, keywords for *m4* [see *m4*(1)] cannot be used as symbols (variables, functions, labels) in the input file since *m4* cannot determine which are assembler symbols and which are real *m4* macros.

BUGS

The **even** assembler directive is not guaranteed to work in the **.text** section when optimization is performed.

Arithmetic expressions may only have one forward referenced symbol per expression.

NAME

mc68conv – MC68000 object file converter

SYNOPSIS

mc68conv [-] [**-a**] [**-o**] [**-p**] [**-s**] **-t** target files

DESCRIPTION

The *mc68conv* command converts object files from their current format to the format of the *target* machine. Mc68conv can read an archive file in any of three formats: the UNIX pre-5.0 format, the 5.0 random access format, and the 6.0 portable ASCII format. It produces a file in the format specified (**-a**, **-o**, or **-p**). The converted file is written to file.*v*.

Command line options are:

- indicates *files* should be read from *stdin*.

-a If the input file is an archive, produces the ouput file in the 6.0 portable ASCII archive format.

-o If the input file is an archive, produces the output file in the UNIX pre-5.0 format.

-p If the input file is an archive, produces the ouput file in the UNIX 5.0 random access archive format. This is the default.

-s Functions exactly as 3bswab, i.e. "preswab" all characters in the object file. This is useful only for AT&T 3B20 Computer object files which are to be "swab-dumped" from a DEC machine to a 3B20 Computer.

-t target Converts the object file to the byte ordering of the machine (*target*) to which the object file is being shipped. This may be another host or a target machine. Legal values for *target* are: pdp, vax, ibm, i80, x86, b16, n3b, m32, and mc68.

Mc68conv can be used to convert all object files in common object file format. It can be used on either the source ("sending") or target ("receiving") machine.

Mc68conv is meant to ease the problems created by a multi-host cross-compilation development environment. Mc68conv is best used within a procedure for shipping object files from one machine to another.

EXAMPLE

```
# ship object files from pdp11 to ibm
$echo *.out ! mc68conv -t ibm -$OFC/foo.o
$uucp *.v my370!~/rje/
```

DIAGNOSTICS

All diagnostics are intended to be self-explanatory. Fatal diagnostics on the command lines cause termination. Fatal diagnostics on an input file cause the program to continue to the next input file.

WARNINGS

Mc68conv will not convert archives from one format to another if both the source and target machines have the same byte ordering. The UNIX tool *convert*(1) should be used for this purpose.

NAME

mc68cpp – the C language preprocessor

SYNOPSIS

$DMD/lib/mc68cpp [option ... **] [** ifile **[** ofile **]]**

DESCRIPTION

Mc68cpp is the C language preprocessor which is invoked as the first pass of any C compilation using the *dmdcc*(1) command. Thus, the output of *mc68cpp* is designed to be in a form acceptable as input to the next pass of the C compiler. As the C language evolves, *mc68cpp* and the rest of the C compilation package will be modified to follow these changes. Therefore, the use of *mc68cpp* other than in this framework is not suggested. The preferred way to invoke *mc68cpp* is through the *dmdcc*(1) command, since the functionality of *mc68cpp* may some day be moved elsewhere. See *m4*(1) for a general macro processor.

Mc68cpp optionally accepts two file names as arguments. *Ifile* and *ofile* are respectively the input and output for the preprocessor. They default to standard input and standard output if not supplied.

The following *options* to *mc68cpp* are recognized:

-P Preprocesses the input without producing the line control information used by the next pass of the C compiler.

-C By default, *mc68cpp* strips C-style comments. If the **-C** option is specified, all comments (except those found on *mc68cpp* directive lines) are passed along.

-U*name*

Removes any initial definition of *name*, where *name* is a reserved symbol that is predefined by the particular preprocessor. The current list of these possibly reserved symbols includes:

operating system:	ibm, gcos, os, tss, unix
hardware:	interdata, pdp11, u370, u3b, u3b5, vax, mc68000, mc68k16, mc68k32
UNIX variant:	RES, RT

-D*name*

-D*name*=*def*

Defines *name* as if by a **#define** directive. If no =*def* is given, *name* is defined as 1.

-I*dir* Changes the algorithm for searching for **#include** files whose names do not begin with / to look in *dir* before looking in the directories on the standard list. Thus, **#include** files whose names are enclosed in " " will be searched for first in the directory of the *ifile* argument, then in directories named in **-I** options, and last in directories on a standard list. For **#include** files whose names are enclosed in <>, the directory of the *ifile* argument is not searched.

Two special names are understood by *mc68cpp*. The name **__LINE__** is defined as the current line number (as a decimal integer) as known by *mc68cpp*, and **__FILE__** is defined as the current file name (as a C string) as known by *mc68cpp*. They can be used anywhere (including in macros) just as any other defined name.

All *mc68cpp* directives start with lines whose first character is **#**. The directives are:

#define *name token-string*
Replaces subsequent instances of *name* with *token-string*.

#define *name*(*arg, ..., arg*) *token-string*
Notice that there can be no space between *name* and the (. Replaces subsequent instances of *name* followed by a (, a list of comma separated tokens, and a) by *token-string* where each occurrence of an *arg* in the *token-string* is replaced by the corresponding token in the comma separated list.

#undef *name*
Causes the definition of *name* (if any) to be forgotten from now on.

#include "*filename*"
#include <*filename*>
Include at this point the contents of *filename* (which will then be run through *mc68cpp*). When the <*filename*> notation is used, *filename* is only searched for in the standard places. See the **-I** option above for more detail.

#line *integer-constant* "*filename*"
Causes *mc68cpp* to generate line control information for the next pass of the C compiler. *Integer-constant* is the line number of the next line and *filename* is the file where it comes from. If "filename" is not given, the current file name is unchanged.

#endif
Ends a section of lines begun by a test directive (**#if**, **#ifdef**, or **#ifndef**). Each test directive must have a matching **#endif**.

#ifdef *name*

The lines following will appear in the output if, and only if, *name* has been the subject of a previous **#define** without being the subject of an intervening **#undef**.

#ifndef *name*

The lines following will not appear in the output if, and only if, *name* has been the subject of a previous **#define** without being the subject of an intervening **#undef**.

#if *constant-expression*

Lines following will appear in the output if, and only if, the *constant-expression* evaluates to non-zero. All binary non-assignment C operators, the **?:** operator, the unary **-**, **!**, and **~** operators are all legal in *constant-expression*. The precedence of the operators is the same as defined by the C language. There is also a unary operator **defined**, which can be used in *constant-expression* in these two forms: **defined (** *name* **)** or **defined** *name*. This allows the utility of **#ifdef** and **#ifndef** in a **#if** directive. Only these operators, integer constants, and names which are known by *mc68cpp* should be used in *constant-expression*. In particular, the **sizeof** operator is not available.

#else Reverses the notion of the test directive which matches this directive. So if lines previous to this directive are ignored, the following lines will appear in the output, and vice versa.

The test directives and the possible **#else** directives can be nested.

FILES

/usr/include standard directory for **#include** files

SEE ALSO

dmdcc(1).
m4(1) in the *UNIX System V User's Reference Manual.*

DIAGNOSTICS

The error messages produced by *mc68cpp* are intended to be self-explanatory. The line number and filename where the error occurred are printed along with the diagnostic.

WARNING

When newline characters were found in argument lists for macros to be expanded, previous versions of *mc68cpp* put out the newlines as they were found and expanded. The current version of *mc68cpp* replaces these newlines with blanks to alleviate problems that the previous versions had when this occurred.

NAME

mc68cprs – compress a MC68000 object file

SYNOPSIS

mc68cprs [**-pv**] infile outfile

DESCRIPTION

The *mc68cprs* command reduces the size of a Motorola 68000 object file, *infile*, by removing duplicate structure and union descriptors. The reduced file, *outfile*, is produced as output.

The options are:

-p Prints statistical messages including:

total number of tags
total duplicate tags
total reduction of *infile*.

-v Prints verbose error messages if error condition occurs.

EXAMPLE

```
mc68cprs dmda.out sm3b
```

SEE ALSO

mc68strip(1).

NAME

mc68dis - MC68000 disassembler

SYNOPSIS

mc68dis [**-o**] [**-V**] [**-L**] [**-d** sec] [**-da** sec] [**-F** function] [**-t** sec] [**-l** string] files

DESCRIPTION

The *mc68dis* command produces an assembly language listing of each of its object *file* arguments. The listing includes assembly statements and a hexadecimal or octal representation of the binary that produced those statements.

The following *options* are interpreted by the disassembler and may be specified in any order.

-o Prints numbers in octal. Default is hexadecimal.

-V Version number of the disassembler is written to standard error.

-L Invokes a look-up of C source labels in the symbol table for subsequent printing.

-d sec Disassembles the named section as data, printing the offset of the data from the beginning of the section.

-da sec Disassembles the named section as data, printing the actual address of the data.

-F function Disassembles the named function in each object file that is specified on the command line.

-t sec Disassembles the named section as text.

-l string Disassembles the library file specified as *string*. For example, one would issue the command **mc68dis -l x -l z** to disassemble **libx.a** and **libz.a**. All libraries are assumed to be in **$DMD/lib**.

If the **-d**, **-da** or **-t** options are specified, only those named sections from each user supplied file name are disassembled. Otherwise, all sections containing text will be disassembled.

If the **-F** option is specified, only those named functions from each user supplied file name are disassembled. **-F** only works with object files that have been compiled with the **dmdcc -g** option.

On output, a number enclosed in brackets at the beginning of a line, such as **[5]**, represents a C break-pointable line number that starts with the following instruction. These line numbers are present only when the object file has been compiled with the **dmdcc -g** option. An expression such as **<40>** in the operand field, following a relative displacement for control transfer instructions, is the computed address within the section to which control is transferred. Similarly, an expression such as **<40>+%d0**, following a program counter index plus displacement operand, indicates that the effective address of the operand in the current section is 40 plus the content of %d0. A C function name will appear in the first column, followed by **()**, if the function was compiled with **-g**.

SEE ALSO

dmdcc(1), mc68as(1), mc68ld(1).

DIAGNOSTICS

The self-explanatory diagnostics indicate errors in the command line or problems encountered with the specified files.

NAME

mc68dump – dump parts of an MC68000 object file

SYNOPSIS

mc68dump [**-acfghlorst**] [**-z** name] files

DESCRIPTION

The *mc68dump* command dumps selected parts of each of its object *file* arguments.

This command accepts both object files and archives of object files. It processes each file argument according to one or more of the following options:

-a Dumps the archive header of each member of each archive file argument.

-g Dumps the global symbols in the symbol table of a 6.0 archive.

-f Dumps each file header.

-o Dumps each optional header.

-h Dumps section headers.

-s Dumps section contents.

-r Dumps relocation information.

-l Dumps line number information.

-t Dumps symbol table entries.

-z name Dumps line number entries for the named function.

-c Dumps the string table.

The following *modifiers* are used in conjunction with the options listed **above to modify their capabilities.**

-d number Dumps the section number or range of sections starting at *number* and ending either at the last section number or *number* specified by **+d**.

+d number Dumps sections in the range either beginning with first section or beginning with section specified by **-d**.

-n name Dumps information pertaining only to the named entity. This *modifier* applies to **-h**, **-s**, **-r**, **-l**, and **-t**.

-p Suppresses printing of the headers.

-t index Dumps only the indexed symbol table entry. The **-t**, used in conjunction with **+t**, specifies a range of symbol table entries.

+t index Dumps the symbol table entries in the range ending with the indexed entry. The range begins at the first symbol table entry or at the entry specified by the **-t** option.

-u Underlines the name of the file for emphasis.

-v Dumps information in symbolic representation rather than numeric (e.g., C_STATIC instead of **0X02**). This *modifier* can be

used with all the above options except **-s** and **-o** options of *mc68dump*.

-z name,number
Dumps line number entry or range of line numbers starting at *number* for the named function.

+z number Dumps line numbers starting at either function *name* or *number* specified by **-z,** up to *number* specified by **+z**.

Blanks separating an *option* and its *modifier* are optional. The comma separating the name from the number modifying the **-z** option may be replaced by a blank.

The **-z** and **-n** options that take a *name* modifier will only work with object files that have been compiled with the **dmdcc -g** option.

The *mc68dump* command attempts to format the information it dumps in a meaningful way, printing certain information in character, hex, octal or decimal representation as appropriate.

SEE ALSO

dmdcc(1).
a.out(4), ar(4) in the *UNIX System V Programer's Reference Manual.*

NAME

mc68ld – link editor for MC68000 object files

SYNOPSIS

mc68ld [options **]** file-names

DESCRIPTION

The *mc68ld* command combines several object files into one, performs relocation, resolves external symbols, and supports symbol table information for symbolic debugging. In the simplest case, the names of several object programs are given, and *mc68ld* combines them, producing an object module that can either be executed or used as input for a subsequent *mc68ld* run. The output of *mc68ld* is left in **mc68a.out**. This file is executable if no errors occurred during the load. If any input file, *file-name*, is not an object file, *mc68ld* assumes it is either a text file containing link editor directives or an archive library. (See the *Link Editor* in the *UNIX System V Support Tools Guide* for a discussion of input directives.)

If any argument is a library, it is searched exactly once at the point it is encountered in the argument list. Only those routines defining an unresolved external reference are loaded. The order of library members is unimportant because *mc68ld* passes through each library's (archive) symbol table as many times as necessary until no new external symbols are resolved and no new references are generated.

The following options are recognized by *mc68ld*.

-a Produces an absolute file; gives warnings for undefined references. Relocation information is stripped from the output object file unless the **-r** option is given. The **-r** option is needed only when an absolute file should retain its relocation information (the normal case for the 630 MTG downloaded programs). If neither **-a** nor **-r** is given, **-a** is assumed.

-e epsym Sets the default entry point address for the output file to be that of the symbol *epsym*.

-f fill Sets the default fill pattern for "holes" within an output section as well as initialized bss sections. The argument *fill* is a two-byte constant.

-l*x* Searches a library **lib***x***.a,** where *x* is up to seven characters. A library is searched when its name is encountered, so the placement of a **-l** is significant. By default, libraries are located in $DMD/lib .

-m Produces a map or listing of the input/output sections on the standard output.

-o outfile Produces an output object file by the name *outfile*. The name of the default object file is **mc68a.out**.

-r Retains relocation entries in the output object file. Relocation entries must be saved if the output file is to become an input file in a subsequent *mc68ld* run. Unless **-a** is also given, the link editor will not complain about unresolved references.

-s Strips line number entries and symbol table information from the output object file.

-u symname Enters *symname* as an undefined symbol in the symbol table. This is useful for loading entirely from a library, since initially the symbol table is empty and an unresolved reference is needed to force the loading of the first routine.

-Ldir Changes the algorithm of searching for **lib***x***.a** to look in *dir* before looking in $DMD/lib. This option is effective only if it precedes the **-l** option on the command line.

-N Puts the data section immediately following the text in the output file.

-V Outputs a message giving information about the version of mc68ld being used.

-VS num Uses *num* as a decimal version stamp identifying the **mc68a.out** file that is produced. The version stamp is stored in the optional header.

-X Generates a standard UNIX file header within the "optional header" field in the output file.

FILES

$DMD/lib/lib?.a	libraries
mc68a.out	output file

SEE ALSO

dmdcc(1), mc68as(1).
a.out(4), ar(4) in the *UNIX System V Programmer's Reference Manual.*

WARNINGS

Through its options and input directives, the Motorola 68000 link editor gives users great flexibility; however, those who use the input directives must assume some added responsibilities. Input directives and options should insure the following properties for programs:

- C defines a zero pointer as null. A pointer to which zero has been assigned must not point to any object. To satisfy this, users must not place any object at virtual address zero in the data space.
- When the link editor is called through *dmdcc*(1), a startup routine is linked with the user's program. This routine usually calls exit( ) [see *exit*(3R)] after execution of the main program. If the user calls the link editor directly, then the user must insure that the program always calls exit( ) rather than falling through the end of the entry routine.

The **-VS** *num* option has an effect only when the **-X** option is also selected.

NAME

mc68lorder – find ordering relation for an object library

SYNOPSIS

mc68lorder file ...

DESCRIPTION

The input is one or more object or library archive *files* [see *mc68ar*(1)]. The standard output is a list of pairs of object file names, meaning that the first file of the pair refers to external identifiers defined in the second file. The output may be processed by *tsort*(1) to find an ordering of a library suitable for one-pass access by *mc68ld*(1). Note that the link editor *mc68ld*(1) is capable of multiple passes over an archive in the portable archive format [see *ar*(4)] and does not require that *mc68lorder*(1) be used when building an archive. The usage of the *mc68lorder*(1) command may, however, allow for a slightly more efficient access of the archive during the link edit process.

The following example builds a new library from existing **.o** files.

```
mc68ar cr library `mc68lorder *.o | tsort`
```

FILES

*symref, *symdef temporary files

SEE ALSO

mc68ar(1), mc68ld(1).
ar(4) in the *UNIX System V Programmer's Reference Manual.*
tsort(1) in the *UNIX System V User Reference Manual.*

BUGS

Object files whose names do not end with **.o**, even when contained in library archives, are overlooked. The global symbols and references are attributed to some other file.

NAME

mc68nm – print name list of a MC68000 object file

SYNOPSIS

mc68nm [options **]** file-names

DESCRIPTION

The *mc68nm* command displays the symbol table of each Motorola 68000 object file *file-name*. *File-name* may be a relocatable or absolute Motorola 68000 object file; or it may be an archive of relocatable or absolute Motorola 68000 object files. For each symbol, the following information is printed. For the **TYPE**, **SIZE**, or **LINE** information, the object file must be compiled with the **-g** option of the *dmdcc*(1) command.

Name The name of the symbol.

Value Its value expressed as an offset or an address depending on its storage class.

Class Its storage class.

Type Its type and derived type. If the symbol is an instance of a structure or of a union, then the structure or union tag is given following the type (e.g. struct-tag). If the symbol is an array, then the array dimensions are given following the type (eg., **char[n][m]**).

Size Its size in bytes, if available.

Line The source line number at which it is defined, if available.

Section For storage classes static and external, the object file section containing the symbol (e.g., text, data or bss).

The output of *mc68nm* may be controlled using the following options:

-d Prints the value and size of a symbol in decimal (the default).

-o Prints the value and size of a symbol in octal instead of decimal.

-x Prints the value and size of a symbol in hexadecimal instead of decimal.

-h Does not display the output header data.

-v Sorts external symbols by value before they are printed.

-n Sorts external symbols by name before they are printed.

-e Prints only external and static symbols.

-f Produces full output. Prints redundant symbols (.text, .data and .bss), that are normally suppressed.

-u Prints undefined symbols only.

-V Prints the version of the mc68nm command executing on the standard error output.

-T By default, *mc68nm* prints the entire name of the symbols listed. Since object files can have symbol names with an arbitrary number of characters, a name that is longer than the width of the column set aside for names will overflow its column, forcing every column after the name to be misaligned. The **-T** option causes *mc68nm* to truncate every name which would otherwise overflow its column and place an asterisk as the last character in the displayed name to mark it as truncated.

Options may be used in any order, either singly or in combination, and may appear anywhere in the command line. Therefore, both **mc68nm name -e -v** and **mc68nm -ve name** print the static and external symbols in *name*, with external symbols sorted by value.

FILES

/usr/tmp/nm??????

SEE ALSO

dmdcc(1), mc68as(1), mc68ld(1).
a.out(4), ar(4) in the *UNIX System V Programmer's Reference Manual.*

DIAGNOSTICS

"mc68nm: name: cannot open"
if *name* cannot be read.

"mc68nm: name: bad magic"
if *name* is not an appropriate Motorola 68000 object file.

"mc68nm: name: no symbols"
if the symbols have been stripped from *name*.

WARNINGS

When all the symbols are printed, they must be printed in the order they appear in the symbol table in order to preserve scoping information. Therefore, the **-v** and **-n** options should be used only in conjunction with the **-e** option.

NAME

mc68size – print section sizes of MC68000 object files

SYNOPSIS

mc68size [**-o**] [**-x**] [**-V**] files

DESCRIPTION

The *mc68size* command produces section size information for each section in the Motorola 68000 object files. The size of the text, data, and bss (uninitialized data) sections are printed along with the total size of the object file. If an archive file is input to the mc68size command the information for all archive members is displayed.

Numbers are printed in decimal unless either the **-o** or the **-x** option is used, in which case they are printed in octal, or in hexadecimal, respectively.

The **-V** flag will supply the version information on the *mc68size* command.

SEE ALSO

dmdcc(1), mc68as(1), mc68ld(1).
a.out(4), ar(4) in the *UNIX System V Programmer's Reference Manual.*

DIAGNOSTICS

"mc68size: name: cannot open"
if *name* cannot be read.

"mc68size: name: bad magic"
if *name* is not a Motorola 68000 object file.

NAME

mc68strip – strip symbolic information from MC68000 object file

SYNOPSIS

mc68strip [**-l**] [**-m**] [**-x**] [**-r**] [**-s**] [**-V**] file-names

DESCRIPTION

The *mc68strip* command strips the symbol table and line number information from Motorola 68000 object files, including archives. Once this has been done, no symbolic debugging access is available for that file; therefore, this command is normally run only on production modules that have been debugged and tested.

The amount of information stripped from the symbol table can be controlled by using the following options:

-l Strips line number information only; does not strip any symbol table information.

-m Strips symbol table information only; does not strip any relocation information. Used for 630 MTG applications which need relocation information for downloads, but do not necessarily need symbol table information. This option does not work on archives.

-x Does not strip static or external symbol information.

-r Resets the relocation indices into the symbol table.

-s Resets the line number indices into the symbol table (does not remove). Resets the relocation indices into the symbol table.

-V Prints the version of the mc68strip command executing on the standard error output.

If there are any relocation entries in the object file and any symbol table information is to be stripped, *mc68strip* will complain and terminate without stripping *file-name* unless the **-r** or **-m** flags are used.

If the *mc68strip* command is executed on a common archive file [see *ar*(4)] the archive symbol table will be removed. The archive symbol table must be restored by executing the *mc68ar*(1) command with the **s** option before the archive can be link edited by the *mc68ld*(1) command. *Mc68strip*(1) will instruct the user with appropriate warning messages when this situation arises.

The purpose of this command is to reduce the file storage overhead taken by the object file.

FILES

/usr/tmp/mc68str??????

SEE ALSO

dmdcc(1), mc68ar(1), mc68as(1), mc68ld(1).
a.out(4), ar(4) in the *UNIX System V Programmer's Reference Manual.*

DIAGNOSTICS

mc68strip: name: cannot open
if *name* cannot be read.

mc68strip: name: bad magic
if *name* is not a Motorola 68000 object file.

mc68strip: name: relocation entries present; cannot strip
if *name* contains relocation entries and the **-r** or **-m** flag is not used, the symbol table information cannot be stripped.

mc68strip: name: other options set with "m" option
if other flags are used with the -m option which is mutually exclusive.

mc68strip: "m" option not allowed on archive files
if file name is an archive file.

NAME

ucache – List and remove objects in the Application cache

SYNOPSIS

ucache

DESCRIPTION

After being downloaded, the program *ucache* will cache itself as a global command and exits. The user can access the command through the **More** submenu under the item *ucache* displayed with a garbage can icon. An arrow from the item directs the user to a submenu that lists sequentially all objects (commands and applications) that are currently in the terminal's Application cache.

The names listed in the *ucache* submenu are the *menu names* of the objects. Menu names are names that cached applications use to advertise themselves in the **More** menu. They may be different from *tag names* which uniquely identify an object in the Application cache. (For further discussion, see *cache(3L)*.) If an object in the Application cache does not have a menu name, its tag name is displayed instead.

Items in the *ucache* submenu may be greyed. Greyed items mean they cannot be selected for removal because:

— they have been cached with the **A_PERMANENT** bit set (see *cache(3L)*). In this case, the item names are displayed with a lock icon, and they can never be uncached by *ucache*.

or

— they have been requested (used) by some running process. If all processes using the object exit or are deleted, then the item's name corresponding to the object will be un-greyed.

Items that are not greyed can be selected for removal. All caching information concerning the object will be erased and the memory freed. Note that *ucache* can remove itself from the Application cache.

SEE ALSO

cache(3L), cmdcache(3L), decache(3L).

NAME

wtinit - initialize 630 MTG terminal for layers environment

SYNOPSIS

wtinit

DESCRIPTION

Wtinit is used by the UNIX layers(1) command to initialize the 630 MTG terminal for the layers environment. This is the 630 MTG terminal specific version of the UNIX windowing utilities *wtinit* command. Layers(1) will use $DMD/bin/wtinit if the $DMD variable is set.

DIAGNOSTICS

Wtinit is a shell program which uses the *dmdversion*(1) command to determine if LAN encoding is set. If *dmdversion* is not found a message is displayed.

EXIT STATUS

Returns 0 upon successful completion, 1 otherwise.

SEE ALSO

layers(1) in the *UNIX System V Release 3 User's Reference Manual.*
wtinit(1) in the *UNIX System V Release 3 System Administrator's Reference Manual.*

NAME

abs – return integer absolute value

SYNOPSIS

int abs (i)
int i;

DESCRIPTION

abs returns the absolute value of its integer operand.

SEE ALSO

floor(3M).

WARNING

In two's-complement representation, the absolute value of the negative integer with largest magnitude is undefined.

NAME

addr – return the Word address of a Point in a Bitmap

SYNOPSIS

#include <dmd.h>

Word *addr (b, p)
Bitmap *b;
Point p;

DESCRIPTION

The *addr* function returns the address of the Word containing the bit corresponding to the Point *p* in the Bitmap *b*.

EXAMPLE

The following subroutine can be used to determine whether a Point *p* in a Bitmap *b* is on or off (returning 1 or 0, respectively).

```
#include <dmd.h>
Word *addr();

pixel (b, p)
Bitmap *b;
Point p;
{
        Word *w;
        UWord bit;

        w = addr (b, p);
        bit = FIRSTBIT >> (p.x%WORDSIZE);
        return (*w&bit)==bit;
}
```

This routine is implemented differently in *rol*(3L).

SEE ALSO

rol(3L), structures(3R).

NAME

alloc, lalloc, free, allocown – memory allocation

SYNOPSIS

```
#include <dmd.h>
char *alloc (nbytes)

char *lalloc (lnbytes)

void free (s)

void allocown (s, p)

unsigned nbytes;
unsigned long lnbytes;
Proc *p;
char *s;
```

DESCRIPTION

The *alloc* function is equivalent to the standard C function *malloc*(3C). It either returns a pointer to a block of *nbytes*, contiguous bytes of storage or a 0 (NULL) if there is no available memory. The storage is aligned on 4-byte boundaries. Unlike *malloc*, *alloc*, clears the storage to zeros.

The *lalloc* function is identical to *alloc* except it takes an unsigned long as an argument.

The *free* function frees storage allocated by *alloc*. The space is made available for further allocation.

The *allocown* function changes the ownership of memory allocated by *alloc*. The argument *s* is a value previously returned from *alloc*. The argument *p* is the process taking over ownership of the memory. If *p* is zero, the memory belongs to no one, and only an explicit call to *free* will free it.

The terminal automatically frees all memory allocated by a process when the process terminates or when the window it is running in is deleted. If the ownership of allocated memory is changed, it will only be freed when the new owner is deleted or when *free* is called. However, it is recommended that a process free its allocated memory when the storage is no longer needed so that other processes will be able to use it.

EXAMPLE

The following example shows the use of *alloc* and *free* in dynamically allocating memory for a *Point*.

```
#include <dmd.h>

main()
{
    Point *p;
    char *alloc();
    void free();

    p = (Point *)alloc (sizeof(Point));
```

```
        .
        .
        .

        free (p);
    }
```

SEE ALSO

balloc(3R), gcalloc(3R), structures(3R).
malloc(3C) in the *UNIX System V Programmer's Reference Manual.*

WARNINGS

The *alloc* function accepts an integer as argument; therefore, it can only allocate a contiguous block of memory of 64K bytes or less.

NAME

atof – convert string to double-precision number

SYNOPSIS

double atof (str)
char *str;

DESCRIPTION

atof returns (as a double-precision floating-point number) the value represented by the character string pointed to by *str*. The string is scanned up to the first unrecognized character.

atof recognizes an optional string of "white-space" characters [as defined by *isspace* in *ctype*(3L)], then an optional sign, then a string of digits optionally containing a decimal point, then an optional **e** or **E** followed by an optional sign or space, followed by an integer.

SEE ALSO

ctype(3L), strtol(3L).

DIAGNOSTICS

If the correct value would cause overflow, ±HUGE (as defined in <**ccs/math.h**>) is returned (according to the sign of the value), and *errno* is set to ERANGE.

If the correct value would cause underflow, zero is returned and *errno* is set to ERANGE.

NAME

attach – connect process to host

SYNOPSIS

int attach (host)
int host;

DESCRIPTION

A process that is already local may connect itself to a host by calling the function *attach*. It takes a single argument indicating the host to be connected to. There are currently only two valid values, 0 and 1, that refer to logical Host 1 and Host 2, respectively. When a process is successfully attached, the border for the window of that process becomes solid.

The resources owned by the process remain unchanged except for the addition of a host connection. The *attach* function operates correctly independent of whether the host is already in **layers** mode or not.

The function can fail if the host has not been configured in the set-up options; if there are not any available connections for the specified host; if the process is already connected; or if the host argument is invalid. A failure is indicated to the calling process by a return value of zero.

EXAMPLE

This example shows how the *attach* function may be used in a process that wishes to be connected to the logical Host 2.

```
#include <dmd.h>


switchhost ()
{
        if (local())
                attach(1);

}
```

SEE ALSO

local(3R), peel(3R).

WARNING

The host values may change in meaning and/or be expanded in the future.

NAME

balloc, bfree – bitmap allocation

SYNOPSIS

#include <dmd.h>

Bitmap *balloc (r)
Rectangle r;

void bfree (b)
Bitmap *b;

DESCRIPTION

The *balloc* function either returns a pointer to a Bitmap large enough to contain the Rectangle *r*, or a 0 (NULL) for failure. The bitmap is allocated first by a call to *alloc(3R)* for the Bitmap data structure and then to *gcalloc(3R)* for the actual Bitmap storage. The coordinate system inside the Bitmap is set by *r*. The origin and corner of the Bitmap are those of *r* which must itself be in screen coordinates.

The *bfree* frees the storage associated with a Bitmap allocated by *balloc*.

The terminal automatically frees all memory *balloc*'ed by a process when the process terminates or when the window it is running in is deleted. However, it is recommended that a process free its *balloc*'ed memory when the storage is no longer needed so that other processes will be able to use it.

EXAMPLE

The following example shows the use of *balloc* and *bfree* in dynamically allocating memory for a Bitmap.

```
#include <dmd.h>

main()
{
    Bitmap *b;
    Bitmap *balloc();
    void bfree();

    b = balloc (Rect (0, 0, 48, 48));
    .
    .
    .
    bfree (b);
}
```

SEE ALSO

alloc(3R), gcalloc(3R), structures(3R).

BUGS

If the bitmap image requested needs over 7000000 bytes, *balloc* may produce a process exception. If this happens, *gcalloc* memory will be corrupted, and other programs running in the terminal may be damaged.

NAME

bitblt – bit-block transfer

SYNOPSIS

#include <dmd.h>

void bitblt (sb, r, db, p, f)
Bitmap *sb, *db;
Rectangle r;
Point p;
Code f;

DESCRIPTION

The *bitblt* function copies the data from Rectangle *r* in Bitmap *sb* to the congruent Rectangle with origin *p* in Bitmap *db*. Copy is specified by the function Code *f*.

The source and destination Bitmaps may be the same or different and the source and destination Rectangles may even overlap; *bitblt* always does the assignments in the correct order.

EXAMPLES

The following subroutine paints a mouse icon into the upper left corner of the applications window.

```
unsigned short mouseicon[] = {
        0x0000, 0x0000, 0x03E0, 0x17F0,
        0x3FF0, 0x5FFE, 0xFFF1, 0x0421,
        0x0002, 0x00FC, 0x0100, 0x0080,
        0x0040, 0x0080, 0x0000, 0x0000,
};

Bitmap mousemap = {
      (Word *)mouseicon,
      1,
      (short)0, (short)0, (short)16, (short)16,
      (char *)0
};

paintmouse()
{
      bitblt(&mousemap, mousemap.rect, &display,
               Drect.origin, F_XOR);
}
```

The following subroutine paints the character *c* of font **ffont* into the upper left corner of the applications window. This is similar to the library function *string*, which paints strings of characters.

```
#include <font.h>

character(ffont, c)
Font *ffont;
char c;
{
        Fontchar *fchar;
        Rectangle r;
        Rectangle fRect();

        fchar = ffont->info + c;
        r = fRect(fchar->x, 0, (fchar+1)->x,
                ffont->height);
        bitblt(ffont->bits, r, &display,
                Drect.origin, F_STORE);
}
```

The following subroutine scrolls a Rectangle *r* in a Bitmap **b* by *n* pixels.

```
scroll(b, r, n)
Bitmap *b;
Rectangle r;
{
      Rectangle s;

      s = r;
      s.origin.y += n; /* scroll up */
      bitblt (b, s, b, r.origin, F_STORE);
      s.origin.y = r.corner.y - n; /* clear bottom */
      rectf (b, s, F_CLR);
}
```

SEE ALSO

structures(3R), string(3R).

NAME

box – draw a Rectangle

SYNOPSIS

#include <dmd.h>

void box (bp, r, f)
Bitmap *bp;
Rectangle r;
Code f;

DESCRIPTION

The *box* function draws the Rectangle *r* in the Bitmap *bp* using function Code *f*. The coordinates of the corner point of the Rectangle *r* are decremented by one before the outline is drawn so that abutting Rectangles do not have common edges.

EXAMPLE

The following subroutine can be used to draw a rectangle in the window and have its position follow mouse movement.

```
#include <dmd.h>

int request();
int wait();
Rectangle raddp();
void box();
int eqpt();

trackRect(tr)
Rectangle tr;
{
      Rectangle r;
      Point     p,q;
      int half_height, half_width;

      half_width  = (tr.corner.x - tr.origin.x)/2;
      half_height  = (tr.corner.y - tr.origin.y)/2;
      request (MOUSE);
      while (!button1())
            wait(MOUSE);
      q = mouse.xy;
      /*
       * position the rectangle with mouse position
       * in the middle
       */
      r = raddp (tr, Pt(q.x - half_width,
               q.y - half_height));
      box (&display, r, F_XOR);
      do {
            /*
             * NOTE:  This do-while loop does not
```

```
                        * give up the CPU for other processes
                        * to run.
                        */
                       p = mouse.xy;
                       if (!eqpt(p, q)){
                       /* allow update of mouse and video
                        * refresh
                        */
                       nap(2);
                       box (&display, r, F_XOR);
                       r = raddp (tr, Pt(p.x - half_width,
                                p.y - half_height));
                             nap(2);
                             box (&display, r, F_XOR);
                             q = p;
                        }
                  } while( bttn(1) );
            }
```

SEE ALSO

structures(3R).

NAME

bputchar – 630 MTG debugging putchar function

SYNOPSIS

void bputchar (c)
char c;

DESCRIPTION

Bputchar is syntactically equivalent to the UNIX standard I/O putchar function. It can be called by downloaded application programs who want to display characters but do not want them displayed in their window. When first called, the bottom third of the 630 MTG's screen is cleared. All characters to be printed are displayed in this area as if it was a window. This will corrupt any windows already in this area. Therefore, this routine is only useful for debugging.

How characters are eventually displayed on a user's terminal when using the UNIX putchar function is affected by the UNIX host *stty*(1) settings and the user's terminal characteristics. Since *bputchar* displays directly onto the 630 MTG screen, it makes assumptions about desired *stty* settings. In general, *bputchar* does as little processing of the output stream as practical.

The following are the only control characters processed by *bputchar*. All other characters will be displayed as **ASCII** characters:

\r Carriage Return. Move the current point to the left edge of the screen.

\n Newline. Move the current point down one line and to the left edge of the window. Scroll the display area if necessary.

\t Horizontal tab. Tab characters are expanded to spaces with tab stops at every eighth space.

EXAMPLE

If a programmer wanted a record of every character it sends to the printer, he can replace every call to *psendchar* with *dpsendchar* shown below.

```
int psendchar();

int
dpsendchar(c)
char c;
{
        int retval;

        retval = psendchar(c);
        if(retval)
                bputchar(c);
        return(retval);
}
```

SEE ALSO

lputchar(3L), printf(3L), psendchar(3R).
putc(3S) in the *UNIX System V Programmer's Reference Manual.*

NAME

bsearch – binary search a sorted table

SYNOPSIS

#include <ccs/search.h>

char *bsearch ((char *) key, (char *) base, nel, sizeof (*key), compar)
unsigned nel;
int (*compar)( );

DESCRIPTION

bsearch is a binary search routine generalized from Knuth (6.2.1) Algorithm B. It returns a pointer into a table indicating where a datum may be found. The table must be previously sorted in increasing order according to a provided comparison function. *Key* points to a datum instance to be sought in the table. *Base* points to the element at the base of the table. *Nel* is the number of elements in the table. *Compar* is the name of the comparison function, which is called with two arguments that point to the elements being compared. The function must return an integer less than, equal to, or greater than zero as accordingly the first argument is to be considered less than, equal to, or greater than the second.

EXAMPLE

The example below searches a table containing pointers to nodes consisting of a string and its length. The table is ordered alphabetically on the string in the node pointed to by each entry.

This code fragment reads in strings and either finds the corresponding node and prints out the string and its length, or prints an error message.

```
#include  <dmd.h>
#include  <dmdio.h>
#include  <ccs/search.h>

#define     TABSIZE      1000

struct node {          /* stored in the table */
      char *string;
      int length;
};
struct node table[TABSIZE]; /* table searched */
      .
      .
      .
{
      struct node *node_ptr, node;
      int node_compare( ); /* compares 2 nodes */
      char str_space[20]; /* to read string into */
      .
      .
      .
      node.string = str_space;
      while (gets(node.string) != NULL) {
```

```
        node_ptr = (struct node *)bsearch(
                   (char *)(&node),
                   (char *)table, TABSIZE,
                   sizeof(struct node),
                   node_compare);
        if (node_ptr != NULL) {
          printf("string = %s, length = %d\n",
                   node_ptr->string,
                   node_ptr->length);
        } else {
          printf("not found: %s\n", node.string);
        }
    }
}
/*
   This routine compares two nodes based on an
   alphabetical ordering of the string field.
*/
int
node_compare(node1, node2)
char *node1, *node2;
{
    return (strcmp(
              ((struct node *)node1)->string,
              ((struct node *)node2)->string));
}
```

Notes

The pointers to the key and the element at the base of the table should be of type pointer-to-element, and cast to type pointer-to-character.

The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

Although *bsearch* is declared as type pointer-to-character, the value returned should be cast into type pointer-to-element.

SEE ALSO

lsearch(3L), qsort(3L).

DIAGNOSTICS

A NULL pointer is returned if the key cannot be found in the table.

BTOC(3R)

NAME

btoc: setjwin, P->btoc, P->ctob - specify rows and columns and default outline

SYNOPSIS

```
#include <dmd.h>

void setjwin (cols, rows)
int cols, rows;

Point (*P->btoc)(x, y, p)
int x,y;
struct Proc *p;

Point (*P->ctob)(x, y, p)
int x,y;
struct Proc *p;
```

DESCRIPTION

The purpose of the *setjwin* and *btoc* functions is to report the number of character rows and columns available in an application's window. These routines are intended to be used by terminal emulator applications executing in the 630 MTG. If the application is running in a *layers* window, the character rows and columns information is sent to the host. This information is then made available to application programs running on the host through an *ioctl*() call to the xt device driver with the *request* argument of JWINSIZE. The windowing utilities *jwin* program is a simple host application which uses this facility to print rows and columns on its standard output. Other host applications, such as the *vi* editor, also use this facility to determine available rows and columns.

The *setjwin* function is called directly from an application program when the application begins execution, either after being downloaded or started from the application cache. The parameters *cols* and *rows* correspond to character columns and character rows respectively.

Btoc is called indirectly by the 630 MTG system control process whenever the application's window is reshaped. *P->btoc* is a pointer stored in the application's process table. The application's process table is simply a structure of type *Proc* that contains system information about the application program and is maintained and used by the 630 MTG's system processes. *P->btoc* points to a function named, for example, *bits_to_char*(), which returns a *Point* structure. The function *bits_to_char*() must be specified in the application program and the *P->btoc* pointer must be set in the application's initialization routine to point to these functions. The *bits_to_char*() function will then be called by the 630 MTG's system control process whenever the application's window is reshaped.

The parameters passed to the *btoc*() function are:

x = the width of the application's window in pixels
y = the height of the application's window in pixels
p = a pointer to the application's process table.

The two integers x and y (returned in the *Point* structure) are the character rows and columns, respectively.

Setjwin and the *btoc*() functions serve similar purposes, but both functions are necessary for the following reasons. *Setjwin* is used to inform the host of character rows and columns when an application boots. The *btoc*() function cannot be used in this situation because the application has not yet executed, and therefore *P->btoc* has not been initialized. On the other hand, the *btoc*() function is used to inform the host of character rows and columns when an application's window is reshaped. The 630 MTG system control process needs to send a message specifying current window size to the host xt driver when a window is reshaped, and it is not possible to wait for the application to call *setjwin*. This makes the *btoc*() function necessary.

If an application does not use the *setjwin* and *btoc* facilities, the host will be told the character row and columns which would be available if Windowproc was running in a window the size of the application's window.

The purpose of a *ctob*() function is to specify a default outline for a window being reshaped based on the *Host* default rows and columns settings specified during the *630 MTG Setup* procedure. See the *630 MTG Terminal* User's Guide for more information about *Setup* and *Host* default rows and columns settings.

Initialization of *P->ctob* is identical to initialization of *P->btoc* as described above. *P->ctob* is set to point to a function named, for example, *char_to_bits*(). The *char_to_bits*() function is called by the 630 MTG system control process before an application's window is reshaped to determine the default window outline to display for the application during the reshape procedure.

The parameters passed to *ctob*() are, in order, x and y (the *Host* default rows and columns, respectively), specified for a window in *Setup*; and *p*, a pointer to the application's process table. The two integers x and y (returned by *ctob*() in the *Point* structure) are the width and height, respectively, in pixels, of the window outline to be displayed.

Note that *ctob* could disregard the parameters x and y and always return a predetermined outline. This is commonly done by non-terminal emulator applications that want to display a default outline which is not necessarily based upon *Setup* options.

If a *ctob* function is not specified, a sweep cursor will appear without a default outline when the application program is reshaped.

The cache(3L) function calls *ctob* to determine the default outline for applications which are invoked from the **More** menu.

EXAMPLE

A simple example of a *bits_to_char*() and a *char_to_bits*() function is shown below.

```
#include <dmd.h>
#include <font.h>

Point bits_to_char();
```

```
Point char_to_bits();

main()
{
        .
        .

    P->btoc = bits_to_char;
    P->ctob = char_to_bits;

        .
        .
}

Point
bits_to_char(x,y,p)
int x,y;
struct Proc *p;
{
    Point q;

    /* INSET is a constant equal to pixel width of */
    /* 630 window border. Defined in dmdproc.h. */
    /* Dmdproc.h is included by dmd.h. */
    q.x = (x - 2*INSET) / FONTWIDTH(largefont);
    q.y = (y - 2*INSET) / FONTHEIGHT(largefont);
    return q;
}

Point
char_to_bits(x,y,p)
int x,y;
struct Proc *p;
{
    Point q;

    q.x = FONTWIDTH(largefont) * x + 2*INSET;
    q.y = FONTHEIGHT(largefont) * y + 2*INSET;
    return q;
}
```

SEE ALSO

cache(3L), globals (3R), structures(3R).
630 MTG Terminal User's Guide.
jwin(1), vi(1) in the *UNIX System V Release 3 User's Reference Manual.*
ioctl(2) in the *UNIX System V Programmer Reference Manual.*
xt(7) in the *UNIX System V Release 3 System Administrator's Reference Manual.*

WARNINGS

Since *btoc*() and *ctob*() are called from the terminal's control process, the variable *P* should not be referenced within these routines. Instead, the parameter *p* should be used to reference the application's process table.

The two integers x and y returned by ctob in the point structure must be less than or equal to XMAX and YMAX, respectively.

NAME

button[123], bttn[123], bttns – button state

SYNOPSIS

```
#include <dmd.h>

int button1 ( ), button2 ( ), button3 ( )
int button12 ( ), button13 ( ), button23 ( ), button123 ( )

int button (b)

int bttn1 ( ), bttn2 ( ), bttn3 ( )
int bttn12 ( ), bttn13 ( ), bttn23 ( ), bttn123 ( )

int bttn (b)

void bttns (updown)

int b;
int updown;
```

DESCRIPTION

The functions *button1*, *button2*, and *button3* return the state of the associated mouse button. They return a non-zero if the button is depressed, 0 if not.

The *button12* function and the other multi-button functions return a Boolean OR of their states, e.g., true if either button 1 or button 2 is depressed (as opposed to button 1 and button 2).

The *button* function takes as an argument the button number 1, 2, or 3 and returns the state of the button. The process must be current and have possession of the mouse. Furthermore, the mouse must be within the bounds of the window.

The bttn routines operate in the same manner as the corresponding button routines except that they do not clip to the process's window. This means that the calling processs must be current and have possession of the mouse. It is not necessary, however, for the mouse cursor to be inside the process's window.

The ability to detect button transitions outside the window is necessary in applications which have menus that may go outside the window. The button routines should be used in preference to the bttn routines unless there is a specific need to be able to detect button state changes outside of the process's window.

Usage of the bttn routines is restricted to routines which do not release the CPU, because the 630 MTG control process also watches for button transitions outside of the current process's window. The control process is the system process which normally handles button operations when the mouse is not in the selected window. Race conditions would otherwise arise as to whether the application process or the control process should interpret the button's state change.

The *bttn* function takes as an argument the button number 1, 2, or 3 and returns the state of the button. It does not clip to the window.

The *bttns* function is used to determine when the mouse state changes. When *bttns* is called, it "busy loops"; not returning and not releasing the CPU until the mouse state changes. If *updown* is 0, *bttns* "busy loops" until all buttons are released. If *updown* is 1, *bttns* "busy loops" until any button is depressed. If *updown* is not 0 or 1, *bttns* returns immediately.

Note that these functions are only valid when **own()&MOUSE** is true.

EXAMPLE

The following code segment could be written to "doodle" in a window.

```
#include <dmd.h>

main()
{
        request (MOUSE);
        for (;;){
                wait (MOUSE);
                if (button3())
                        break;
                if (button1())
                        point (&display, mouse.xy,
                            F_STORE);
        }
}
```

SEE ALSO

resources(3R), transform(3R/3L).

NAME

cache – put the calling application into the Application cache

SYNOPSIS

```
#include <dmd.h>
#include <object.h>

int cache (s, f)
char *s;
int f;
```

DESCRIPTION

An application can put itself into the terminal's Application cache by calling the function *cache*. When this has been done, the window the application is running in can be deleted by the user or the application can exit, but the application itself is still resident in the terminal's memory.

There are two ways to bring up a cached application. If the *menu name* of the application shows up under the **More** menu, the user can select it and open a new window to run the application. Otherwise, an application can be booted from a window running a 630 terminal emulator by calling *dmdld(1)* with the *tag name* of the application. The cached application then replaces the terminal emulator in the window.

The *tag name* of a cached application is the filename it is downloaded under (i.e. *argv[0]*) stripped from any pathname prefix. For example the tag name of *$HOME/dmda.out* is *dmda.out*. This name is used to uniquely identify a cached application.

The *menu name* of a cached application is the name as appeared on the **More** menu. If the argument *s* is a null pointer, the default menu name is the same as the tag name. If *s* is initialized to some character string, that string will be the menu name of the cached application.

The programmer can also specify how an application is cached and how it will be subsequently invoked through the bit-vector argument *f* by OR'ing these constant flags:

A_SHARED

The application is cached as a shared application. A shared application can have multiple copies of it running at the same time. All initialized or un-initialized global and static variables of the application are also shared between all the running copies, so shared applications must be written accordingly. By definition the **A_SHARED** flag forces the **A_DATA** and **A_BSS** flags.

A_NO_SHOW

The application does not want to advertise itself through the **More** menu. Usually this type of application requires host support to run, thus locally opening a window through the **More** menu is not sufficient. In this case, it is preferable to let the host side boot the terminal side (i.e., the cached application) with *dmdld(1)*.

A_BSS The application does not want its un-initialized global and static variables (*.bss* section) to be reset to null for subsequent invocations. To conform with the default initialization rule of the C language which states that un-initialized global and static variables are guaranteed to be set to zero, the default for a cached application is to have its un-initialized global and static variables to be cleared of all updates made by previous runs before a new invocation of the application starts to execute. However, for shared applications or special applications that want to keep data accumulated from previous runs, the **A_BSS** flag can be set to prevent the *.bss* section from being cleared.

A_DATA
The application does not want its initialized global and static variables (*.data* section) to be reset to the original values when the function *cache* is called. By default, when the function *cache* is called, an instant snapshot of the *.data* section is made and stored into memory. Whenever the cached application is invoked again, the saved copy is used to re-initialize the *.data* section with the original values. However, for shared applications, or special applications that want to keep data accumulated from previous runs, or applications that do not change the values of the variables in the *.data* section and do not want memory wasted for a snapshot, the **A_DATA** flag can be set to forgo the savings and copy.

NOTE: The original values of the variables in the *.data* section are the values *at the time* the function *cache* is called. If any of these variables are modified before *cache* is called, the values remembered may not be the same as appeared in the source code.

A_NO_BOOT
The application does not want to be booted from *dmdld*. Note that if both **A_NO_BOOT** and **A_NO_SHOW** are set, there is no way to access the cached application for invocation.

A_PERMANENT
The application cannot be removed by *decache(3L)* or *ucache(1)*. ROM-resident applications are cached this way.

The default when the bit-vector argument *f* is null is to cache the application as a non-shared text, which can be accessed from the **More** menu and from *dmdld(1)*, has its *.bss* section cleared and its *.data* initialized before execution, and can be removed from the application cache.

Besides the information supplied by the arguments, caching a downloaded application requires other parameters. The most useful ones are:

- host connection
- capability to reshape
- default window size

The *cache* function extracts the above information from the current disposition of the application itself. This relieves the programmer from supplying the many parameters, and ensures a uniform user interface among different

cached applications.

The state of host connection of a cached application is the same as of the application when the function *cache* is called. If the application is already local (no host connection), the application will be cached as local; otherwise, it will be cached as connected. When using the **More** submenu to create a connected cached application, the user has to select the host he wants the application to be connected to through a Host submenu (like the one under the item **New** in the global menu). On the other hand, when *dmdld* is used to bootstrap a local cached application to replace the default terminal emulator, the previously connected window will automatically loose its connection (i.e. its border is changed to a checkered pattern of a local window).

The capability to reshape a cached application depends on the *NO_RESHAPE* bit of the process state variable when the function *cache* is called. If this bit is set, the application is cached as non-reshapable. When using the **More** submenu to create a non-reshapable cached application, the user gets the application's default window size without a sweep cursor. When *dmdld* is used to bootstrap a non-reshapable cached application to replace the default terminal emulator, the window is automatically reshaped to the application's default window size.

The cached application's default window size is determined by the *char_to_bits* function (see *btoc(3R)*) and the *NO_RESHAPE* bit. If a function *char_to_bits* is defined for the application, the function *cache* will call it with three arguments: 0, 0, and a pointer to the application's process table to calculate the default window size. The result will be stored in the Application cache, and used by the terminal to generate the default outline if the application is selected from the **More** menu, or to reshape the window if the application is not reshapable and is invoked from *dmdld*. If the *char_to_bits* function is not defined, but the *NO_RESHAPE* bit is set, the default window size will be taken as the current window size of the application when the function *cache* is called. If neither the function *char_to_bits* is defined nor the *NO_RESHAPE* bit is set, no default window size will be displayed when the user selects the application's name under the **More** menu, and the user will have to sweep a window to run the application.

All applications that are not cached with the **A_NO_SHOW** bit on, will be shown on the **More** menu. What happens when selected depends on how they are cached, as explained below.

If an application is cached as shared and local, selection of the application's name in the More submenu always results to the creation of a local window running the chosen application.

If an application is cached as shared and connected, there will be always a Host submenu connected to the application's name in the More submenu. Selection will be effective only when an item in the Host submenu is picked. Selection of the application's name in the More submenu is a null operation.

If an application is cached as non-shared and local, selection of the application's name in the More submenu results in the creation of a local window running the application, *if and only if* no other window is running

that application at the time. Otherwise the window running the application will be made Top and Current.

If an application is cached as non-shared and connected, there will be a Host submenu connected to the application's name in the More submenu, if and only if no other window is running that application at the time. Selection is effective only when an item in the Host submenu is picked. If there is a window running the application already, there will be no Host submenu, and selection of the application's name in the More submenu results in the window running the application being made Top and Current.

Return Value

If the calling application is successfully cached, the *cache* function returns a 1. Otherwise a 0 is returned.

A failure may be due to the following reasons. Another application of the same *tag name* is already in the cache, or the terminal runs out of memory when saving the caching information.

EXAMPLE

This is an example of a very crude terminal emulator that only prints what it receives from the host. This program, called *term.c* is compiled as followed:

```
dmdcc -o term term.c
```

in order to have the name *term* in the **More** submenu.

```
#include <dmd.h>
#include <object.h>

main ()
{
    register int c;
    Point setsize();


    P->ctob = setsize;
    cache ((char *)0, A_SHARED);
    /* cache as shared application */

    request (RCV);
    while (1) {          /* never ending loop */
        wait (RCV);     /* wait for a character */
        while ((c = rcvchar()) != -1)
            lprintf ("%c", c);
            /* print anything received */
      }
}

Point
setsize () /* do not need arguments 0, 0, P */
```

```
{
        Point fPt();

        return (fPt(728, 344));    /* just a nice size */
}
```

SEE ALSO

dmdld(1), ucache(1), btoc(3R), cmdcache(3L), decache(3L), local(3R), state(3R).

NAME

canon – return canonical Rectangle format from two corner Points

SYNOPSIS

#include <dmd.h>

Rectangle canon (p1, p2)
Point p1, p2;

DESCRIPTION

The *canon* function returns a Rectangle created from two Points *p1* and *p2* such that:

r.origin.x equals the minimum of *p1.x* and *p2.x*
r.origin.y equals the minimum of *p1.y* and *p2.y*
r.corner.x equals the maximum of *p1.x* and *p2.x*
r.corner.y equals the maximum of *p1.y* and *p2.y*

In other words, the rectangle defined by the two corner points, p1 and p2, is returned in the standard format of (upper left, lower right).

EXAMPLE

Each of the following cases will yield the Rectangle.

```
{ 0, 0, 32, 32}

canon( Pt(0, 32), Pt(32, 0) )

canon( Pt(32, 32), Pt(0, 0) )

canon( Pt(32, 0), Pt(0, 32) )
```

SEE ALSO

structures(3R).

NAME

circle, disc, discture, arc – circle routines

SYNOPSIS

```
#include <dmd.h>

void circle (b, p, r, f)

void disc (b, p, r, f)

void discture (b, p, r, t, f)

void arc (b, p, p1, p2, f)

Bitmap *b;
Point p, p1, p2;
int r;
Texture16 *t;
Code f;
```

DESCRIPTION

The *circle* function draws the best approximate circle of radius *r* centered at Point *p* in the Bitmap *b* with Code *f*. The circle is guaranteed to be symmetrical about the horizontal, vertical, and diagonal axes.

The *disc* function draws a disc of radius *r* centered at Point *p* in the Bitmap *b* with Code *f*. A disc is a circle which has been completely filled.

The *discture* function draws a disc of radius *r* centered at Point *p* in the Bitmap *b* using the Texture16 *t* with Code *f*. The *discture* function is similar to the *disc* function except it allows one to specify a pattern to fill the disc.

The *arc* function draws a circular arc centered on *p*, traveling counterclockwise from *p1* to the point on the circle closest to *p2*.

EXAMPLE

The following routine draws a "smiling face" in the display Bitmap with center specified by clicking button 1.

```
#include <dmd.h>


main()
{
        int radius;
        extern Texture16 T_darkgrey;
        Point s;

        request(MOUSE);
        wait(MOUSE);
        bttns(1);

        s = mouse.xy;
        radius = 50;
```

```
        /* smiling will draw the face.  Nose will be
         * placed wherever the mouse is clicked in
         * the window.
         */

        smiling(&display, s, radius, &T_darkgrey,
        F_XOR);

        request(KBD);
        wait(KBD);
}


smiling(b, c, rad, t, f)
Bitmap  *b;
Point    c;
int      rad;
Texture16 *t;
Code     f;
{
    int mino, e; /* offsets for placing */
                 /* eyes, nose and mouth */
    int enrad;   /* radius of eyes and nose */

    mino = rad/2;
    e = rad/3;
    enrad = e/3;
    circle (b, c, rad, f); /* face outline */
    disc (b, Pt(c.x-e, c.y-mino),
         enrad, f);        /* left eye */
    disc (b, Pt(c.x+e, c.y-mino),
         enrad, f);        /* right eye */
    discture (b, c, enrad, t, f); /* nose */
    arc (b, c, Pt(c.x-mino, c.y+mino),
      Pt(c.x+mino, c.y+mino),f);
     /* mouth */
}
```

SEE ALSO

ellipse(3L), jcircle(3L), jellipse(3L), texture(3R).

CMDCACHE(3L)

NAME

cmdcache, useritems – cache a command in the Application cache

SYNOPSIS

```
#include <dmd.h>
#include <menu.h>
#include <object.h>

Titem1 useritems;

int cmdcache (s, m, b, u, e)
char *s;
Tmenu *m;
Bitmap *b;
void (*u)( );
void (*e)( );
```

DESCRIPTION

The *cmdcache* function allows the caching of an application as a command to expand the basic set of global menu commands accessed from button 3, such as **New**, **Reshape**, etc.. Although the basic set of commands mostly deals with window operations, a cached command can have other functionalities. The criteria to decide if an application should be cached as a command or as an application is as follows:

The application runs without a window.

The application is compact and specialized, and its user interface must be done through the mouse.

The scope of the application is global to the whole terminal.

An example of a cached command is the terminal resident Exit command.

The application format is just a vehicle to download the code of the command into the terminal; therefore, it should use a template similar to the one used in the **EXAMPLE** section, which has *main()* that calls the *cmdcache* function with the right parameters and exits immediately.

The *cmdcache* function will use the null terminated ASCII string *s* as the *tag* name (see *cache(3L)* for a definition of a *tag* name), and also as the *menu* name of the command. The menu name will be displayed on the **More** menu for user selection. Note that a command can only be accessed from the **More** menu, and booting it through *dmdld* will not work.

The bitmap *b*, if not null, will be displayed as an icon on the left on the menu name of the command. It is recommended that cached commands have an icon to differentiate them from cached applications, since the **More** menu lists all of them nondiscriminatively.

The cached command can have a submenu of its own if the argument *m* is defined. The submenu *m* has to be generated *dynamically*, and the menu generator must use the globally defined *useritems* (see *tmenuhit(3R)* for details of a dynamic menu generation). The structure for *useritems* is **Titem1**, which is used for all global commands and is defined as follows:

```
typedef struct Titem1 {
        char *text;
        struct {
                unsigned short uval;
                unsigned short grey;
        } ufield;
        struct Tmenu *next;
        Bitmap *icon;
        void (*dfn)();
} Titem1;
```

The field *next* does not apply for items in a cached command's submenu (i.e., the terminal does not support fourth-level global submenus). Since submenu off the item is not supported, the field *dfn* is also not relevant. All other fields can be updated by the command's menu generator.

The argument function *u* is called by the terminal during the generation of the dynamic **More** menu. (The **More** menu keeps changing because applications and commands can constantly cache in and decache out.) When the terminal processes a cached command, it copies the static information of the command into *useritems*: the argument *s* goes into the item field *text*, the argument *b* into item field *icon*, the argument *m* into item field *next*. Then the terminal will call the function of type *void* pointed to by *u*, if present, with two arguments: the first one is a pointer to the command object under consideration (this argument is reserved) , and the other is a pointer to the item structure being initialized (i.e., *useritems*). The function pointed by *u* may dynamically update the item's *ufield.grey*, and also re-initialize any fields of the item structure (e.g., if the current conditions dictate that all items in the command's submenu are invalid, the command may decide not to have a submenu, so the function pointed by *u* may change the field *next* to null).

Note The fields *ufield.uval* and *dfn* are used by the terminal for housekeeping: they differentiate between selections of cached applications, cached commands in the **More** menu and items in third-level submenus of those applications and commands. Even though it is possible to modify them, it is *strongly* not recommended unless the programmer understands the terminal's internals.

If the argument *u* is not specified, the *cmdcache* supplies a default function which initializes the *grey* field of the item to null (no greying).

The argument function *e* of type *void* is called when the command or an item from its submenu *m* is selected. The function *e* accepts two arguments: the first one is -1 if the command does not have a submenu, or the index of the selected item (i.e., *useritems.ufield.uval*) if the command has a submenu. The second argument is the pointer to the cached command object, but it is reserved. The function *e* is executed in the context of the terminal's control process like other global menu commands; therefore, some global variables relating to the downloaded application template and window parameters such as *P*, *display*, *Drect*, etc., do not have any meaning.

If the argument function *e* is not specified, no action is taken if the item is selected. Also if the command has a submenu, selecting the command menu name in the **More** menu instead of an item in the command's submenu will result in a null operation.

Return Value

If the command is successfully cached, the function *cmdcache* returns a 1. Otherwise, a 0 is returned.

A failure may be due to the following reasons. Another command or application of the same name is already in the cache, or the terminal runs out of memory when initiating the caching operation.

EXAMPLE

The following application caches a command which lets the user pick a window and displays the name of the application running in that window. The command supports two options: the **full** option displays the full name, and the **clipped** option displays the full name clipped from any path name prefix. The uargv field of the selected process *p* holds the *argv* used by that process.

Note that if the chosen application is cached, the displayed clipped name is the *tag* name it is cached under.

```
#include <dmd.h>
#include <menu.h>
#include <object.h>

struct Tmenu obmenu;
Word qmarkdata[] = {
        0x3C00,
        0x7E00,
        0xE700,
        0xC300,
        0x0300,
        0x0700,
        0x0E00,
        0x1C00,
        0x1800,
        0x1800,
        0x0000,
        0x1800,
        0x1800
};
Bitmap qmark = {
        (Word *)qmarkdata,
        1,
        0, 0, 8, 13,
        0
};
```

```
main ()
{
    Titem1 *genesis(); /* command's submenu generator */
    void showname(); /* command's executing code */


    obmenu.item = (Titem *)0;       /* dynamic submenu */
    obmenu.generator = (Titem *(*)())genesis;
    obmenu.menumap = TM_TEXT|TM_UFIELD|TM_NEXT|
            TM_ICON|TM_DFN;

    cmdcache ("name", &obmenu, &qmark, 01, showname);
}


Titem1 *
genesis (i, m)
int i;
Tmenu *m;
{
    register Titem1 *item = &useritems;
    /* MUST use "useritems" */

    switch (i) {
            case 0:             /* first item */
                item->text = "full";
                break;
            case 1:
                item->text = "clipped";
                break;
            default:    /* last item */
                item->text = (char *)0;
                return (item);
      }

    item->ufield.uval = i;

    /* WARNING: "useritems" is a global variable
    ** used by all cached commands that have a
    ** submenu, so we cannot assume that fields
    ** that are not initialized by genesis() are
    ** cleared since other
    ** commands may initialize
    ** them when they are running.
    **
    ** To be sure, just clear any unused fields.
    */
    item->ufield.grey = 0;
    item->icon = (Bitmap *)0;
    return (item);      /* returns "useritems" */
```

```
}


void
showname (val)
int val;
{
      register Proc *p;
      register char *s;
      Proc *point2window();
      char *clipprefix();

      p = point2window (3);   /* pick a window */
      s = p->uargv[0];        /* "full" name */
      if (val)
         s = clipprefix(s);     /* "clipped" name */
      msgbox (s, (char *)0); /* display name */
}
```

SEE ALSO

ucache(1), cache(3L), decache(3L), tmenuhit(3R).

NAME

conv: toupper, tolower, _toupper, _tolower, toascii - translate characters

SYNOPSIS

#include <ccs/ctype.h>

int toupper (c)
int c;

int tolower (c)
int c;

int _toupper (c)
int c;

int _tolower (c)
int c;

int toascii (c)
int c;

DESCRIPTION

Toupper and *tolower* have as domain the range of *jx*(1) *getc*: the integers from −1 through 255. If the argument of *toupper* represents a lower-case letter, the result is the corresponding upper-case letter. If the argument of *tolower* represents an upper-case letter, the result is the corresponding lower-case letter. All other arguments in the domain are returned unchanged.

The macros *_toupper* and *_tolower*, are macros that accomplish the same thing as *toupper* and *tolower* but have restricted domains and are faster. *_toupper* requires a lower-case letter as its argument; its result is the corresponding upper-case letter. The macro *_tolower* requires an upper-case letter as its argument; its result is the corresponding lower-case letter. Arguments outside the domain cause undefined results.

Toascii yields its argument with all bits turned off that are not part of a standard ASCII character; it is intended for compatibility with other systems.

SEE ALSO

jx(1), ctype(3L), ascii(5).

NAME

ctype: isdigit, isxdigit, islower, isupper, isalpha, isalnum, isspace, iscntrl, ispunct, isprint, isgraph, isascii – character handling

SYNOPSIS

#include <ccs/ctype.h>

int isdigit (c)
int c;

. . .

isascii (c)
int c;

. . .

DESCRIPTION

The character classification macros listed below return nonzero for true, zero for false. *isascii* is defined on all integer values; the rest are defined on valid members of the character set and on the single value -1 (guaranteed not to be a character set member).

isdigit	tests for the digits 0 through 9.
isxdigit	tests for any character for which *isdigit* is true or for the letters *a* through *f* or *A* through *F*.
islower	tests for any lowercase letter as defined by the character set.
isupper	tests for any uppercase letter as defined by the character set.
isalpha	tests for any character for which *islower* or *isupper* is true and possibly any others as defined by the character set.
isalnum	tests for any character for which *isalpha* or *isdigit* is true.
isspace	tests for a space, horizontal-tab, carriage return, newline, vertical-tab, or form-feed.
iscntrl	tests for "control characters" as defined by the character set.
ispunct	tests for any character other than the ones for which *isalnum*, *iscntrl*, or *isspace* is true or space.
isprint	tests for a space or any character for which *isalnum* or *ispunct* is true or other "printing character" as defined by the character set.
isgraph	tests for any character for which *isprint* is true, except for space.
isascii	tests for an ASCII character (a non-negative number less than 0200.)

All the character classification macros do a table lookup.

SEE ALSO

conv(3L), ascii(5).

DIAGNOSTICS

If the argument to any of the character handling macros is not in the domain of the function, the result is undefined.

NAME

cursor: cursinhibit, cursallow, cursswitch, cursset, cursxyon, cursxyoff, Cursinhibit, Cursallow, Cursswitch – cursor control

SYNOPSIS

#include <dmd.h>

void cursallow ( )
void Cursallow ( )

void cursinhibit ( )
void Cursinhibit ( )

Texture16 *cursswitch (t)
Texture16 *Cursswitch (t)

void cursset (p)

void cursxyon ( )

void cursxyoff ( )

Texture16 *t;
Point p;

Texture16 C_target, C_arrows, C_insert;

Texture16 C_cup, C_deadmouse, C_skull;

DESCRIPTION

The *cursinhibit* function turns off the interrupt-time cursor tracking (the drawing of the cursor on the screen), but the mouse coordinates are still kept current and available in the global structure *mouse*.

The *cursallow* function enables interrupt-time cursor tracking.

The functions *cursallow* and *cursinhibit* stack. To enable cursor tracking after two calls to *cursinhibit*, two calls to *cursallow* are required.

The *cursswitch* function changes the mouse cursor to the Texture16 specified by *t*. If *t* is (Texture16 *)0, the cursor is restored to the default arrow. The *cursswitch* function returns the previous value of the cursor, the argument of the previous call to *cursswitch*.

The *Cursallow*, *Cursinhibit*, and *Cursswitch* functions are the same as those described above, but they do not require ownership of the mouse or that the mouse be in the window.

The *cursset* function moves the mouse cursor from the current screen position to the new screen position at Point *p*.

The *cursxyon* function restricts interrupt-time cursor tracking to only the vertical or horizontal axis. The choice of movement along an axis is determined at interrupt-time and depends on the greater mouse movement along the axes. The lesser movement is ignored. This function is used by tmenuhit to help restrict the mouse's movement.

The *cursxyoff* function restores normal cursor tracking.

All of these functions require the mouse to be requested first. They work on a per process basis. They will not affect the mouse operation of other processes.

The Texture16s listed are resident in the 630 MTG. Their names explain what they look like.

EXAMPLE

The following example divides a window into four Rectangles. Based on which Rectangle the mouse is in, this program either switches the cursor to the default arrow, switches the cursor to the AT&T Logo, inhibits the cursor, or sets the cursor to *Drect.origin*.

```
#include <dmd.h>
Texture16 att = {
        0x07E0, 0x1F08, 0x0000, 0x7FFE,
        0x3FC2, 0x0000, 0xFFFF, 0x7FC1,
        0x0000, 0xFFFF, 0x1F01, 0x0000,
        0x7FFE, 0x0000, 0x1008, 0x07E0,
};

Point div(), sub(), add();

main()
{
    Point o, p;
    Rectangle tl, tr, bl, br;
    int lastr = 0;

    o = div (sub (Drect.corner,
            Drect.origin), 2);
    tl.origin = tr.origin = bl.origin
             = br.origin = Drect.origin;
    tr.origin.x += o.x;
    bl.origin.y += o.y;
    br.origin = add (br.origin, o);
    tl.corner = add (tl.origin, o);
    tr.corner = add (tr.origin, o);
    bl.corner = add (bl.origin, o);
    br.corner = add (br.origin, o);
    request (MOUSE|KBD);
    while( kbdchar() == -1 ){
        wait (MOUSE);
        p = mouse.xy;
        if( ptinrect (p, tl) && lastr!=1 ){
          if( lastr==3) cursallow ();
          lastr = 1;
          cursswitch((Texture16 *)0);
        } else if( ptinrect (p, tr) &&
                  lastr!=2 ){
                if( lastr==3) cursallow ();
```

```
                    lastr = 2;
                    cursswitch (&att);
                } else if( ptinrect (p, bl) &&
                          lastr!=3 ){
                    lastr = 3;
                    cursinhibit ();
                } else if( ptinrect (p, br) &&
                          lastr!=4 ){
                    if( lastr==3) cursallow ();
                    lastr = 4;
                    cursset (Drect.origin);
            }
        }
    }
```

SEE ALSO

resources(3R), sleep(3R), structures(3R).

WARNING

The *Cursallow*, *Cursinhibit*, and *Cursswitch* functions change the state of the mouse cursor without informing the 630 MTG operating system. Any action they do must be undone before calling *sleep* or *wait*. Also, they should not be mixed with the other cursor control routines.

NAME

decache – remove the calling application from the Application cache

SYNOPSIS

int decache ( )

DESCRIPTION

The function *decache* lets the calling application remove itself from the Application cache.

The function *decache* frees up all the system's information used to cache the application but does not delete the application itself; it just returns the memory occupied by the application back to the application, so it can be automatically freed when the application exits or is deleted.

The function *decache* returns a 1 if the operation is successful, a 0 otherwise. Failure can be caused by not finding the calling application in the Application cache, and by finding the cached application is currently in use, or cannot be removed (i.e. see the discussion on **A_PERMANENT** flag in *cache(3L)*).

EXAMPLE

The following program illustrates the relationship between *decache* and *cache(3L)*.

```
#include <dmd.h>
#include <object.h>

main ()
{
        register int n;

        lprintf ("Type c to cache\n");
        lprintf ("Type u to uncache\n");
        lprintf ("Type q to quit");
        request (KBD);
        while (wait(KBD)) {
                n = kbdchar();
                lprintf ("\ncharacter typed: %c", n);
                if (n == 'c') {
                        n = cache ("test", 0);
                        lprintf (" -- cache returns %d", n);
                }
                else if (n == 'u') {
                        n = decache ();
                        lprintf (" -- decache returns %d", n);
                }
                else if (n == 'q')
                        break;
        }
}
```

SEE ALSO
ucache(1), cache(3L), cmdcache(3L).

NAME

drand48, erand48, lrand48, nrand48, mrand48, jrand48, srand48, seed48, lcong48 – generate uniformly distributed pseudo-random numbers

SYNOPSIS

double drand48 ( )

double erand48 (xsubi)
unsigned short xsubi [3];

long lrand48 ( )

long nrand48 (xsubi)
unsigned short xsubi [3];

long mrand48 ( )

long jrand48 (xsubi)
unsigned short xsubi [3];

void srand48 (seedval)
long seedval;

unsigned short *seed48 (seed16v)
unsigned short seed16v[3];

void lcong48 (param)
unsigned short param[7];

DESCRIPTION

This family of functions generates pseudo-random numbers using the well-known linear congruential algorithm and 48-bit integer arithmetic.

Functions *drand48* and *erand48* return non-negative double-precision floating-point values uniformly distributed over the interval [0.0, 1.0).

Functions *lrand48* and *nrand48* return non-negative long integers uniformly distributed over the interval $[0, 2^{31})$.

Functions *mrand48* and *jrand48* return signed long integers uniformly distributed over the interval $[-2^{31}, 2^{31})$.

Functions *srand48, seed48* and *lcong48* are initialization entry points, one of which should be invoked before either *drand48, lrand48* or *mrand48* is called. (Although it is not recommended practice, constant default initializer values will be supplied automatically if *drand48, lrand48* or *mrand48* is called without a prior call to an initialization entry point.) Functions *erand48, nrand48* and *jrand48* do not require an initialization entry point to be called first.

All the routines work by generating a sequence of 48-bit integer values, X_i, according to the linear congruential formula

$$X_{n+1} = (aX_n + c)_{\text{mod } m} \qquad n \geq 0.$$

The parameter $m = 2^{48}$; hence 48-bit integer arithmetic is performed. Unless *lcong48* has been invoked, the multiplier value a and the addend value c are given by

$$a = 5\text{DEECE66D}_{16} = 273673163155_8$$
$$c = \text{B}_{16} = 13_8.$$

The value returned by any of the functions *drand48, erand48, lrand48, nrand48, mrand48* or *jrand48* is computed by first generating the next 48-bit X_i in the sequence. Then the appropriate number of bits, according to the type of data item to be returned, are copied from the high-order (leftmost) bits of X_i and transformed into the returned value.

The functions *drand48, lrand48* and *mrand48* store the last 48-bit X_i generated in an internal buffer, and must be initialized prior to being invoked. The functions *erand48, nrand48* and *jrand48* require the calling program to provide storage for the successive X_i values in the array specified as an argument when the functions are invoked. These routines do not have to be initialized; the calling program must place the desired initial value of X_i into the array and pass it as an argument. By using different arguments, functions *erand48, nrand48* and *jrand48* allow separate modules of a large program to generate several *independent* streams of pseudo-random numbers, i.e., the sequence of numbers in each stream will *not* depend upon how many times the routines have been called to generate numbers for the other streams.

The initializer function *srand48* sets the high-order 32 bits of X_i to the 32 bits contained in its argument. The low-order 16 bits of X_i are set to the arbitrary value 330E_{16}.

The initializer function *seed48* sets the value of X_i to the 48-bit value specified in the argument array. In addition, the previous value of X_i is copied into a 48-bit internal buffer, used only by *seed48*, and a pointer to this buffer is the value returned by *seed48*. This returned pointer, which can just be ignored if not needed, is useful if a program is to be restarted from a given point at some future time — use the pointer to get at and store the last X_i value, and then use this value to reinitialize via *seed48* when the program is restarted.

The initialization function *lcong48* allows the user to specify the initial X_i, the multiplier value a, and the addend value c. Argument array elements *param[0-2]* specify X_i, *param[3-5]* specify the multiplier a, and *param[6]* specifies the 16-bit addend c. After *lcong48* has been called, a subsequent call to either *srand48* or *seed48* will restore the "standard" multiplier and addend values, a and c, specified on the previous page.

SEE ALSO

rand(3L).

NAME

ecvt, fcvt, gcvt – convert floating-point number to string

SYNOPSIS

```
char *ecvt (value, ndigit, decpt, sign)
double value;
int ndigit, *decpt, *sign;

char *fcvt (value, ndigit, decpt, sign)
double value;
int ndigit, *decpt, *sign;

char *gcvt (value, ndigit, buf)
double value;
int ndigit;
char *buf;
```

DESCRIPTION

ecvt converts *value* to a null-terminated string of *ndigit* digits and returns a pointer thereto. The high-order digit is non-zero, unless the value is zero. The low-order digit is rounded. The position of the decimal point relative to the beginning of the string is stored indirectly through *decpt* (negative means to the left of the returned digits). The decimal point is not included in the returned string. If the sign of the result is negative, the word pointed to by *sign* is non-zero, otherwise it is zero.

Fcvt is identical to *ecvt*, except that the correct digit has been rounded for printf "%f" (FORTRAN F-format) output of the number of digits specified by *ndigit*.

Gcvt converts the *value* to a null-terminated string in the array pointed to by *buf* and returns *buf*. *Gcvt* attempts to produce *ndigit* significant digits in FORTRAN F-format if possible; otherwise, E-format that is ready for printing. A minus sign, if there is one, or a decimal point will be included as part of the returned string. Trailing zeros are suppressed.

SEE ALSO

printf(3L).

BUGS

The values returned by *ecvt* and *fcvt* point to a single static data array whose content is overwritten by each call.

NAME

ellipse, eldisc, eldiscture, elarc – draw an ellipse

SYNOPSIS

```
#include <dmd.h>

void ellipse (bp, p, a, b, f)
void eldisc (bp, p, a, b, f)
void eldiscture (bp, p, a, b, t, f)
void elarc (bp, p, a, b, p1, p2, f)
Bitmap *bp;
Point p, p1, p2;
int a, b;
Texture16 *t;
Code f;
```

DESCRIPTION

The *ellipse* function draws an ellipse centered at *p* with horizontal semi-axis *a* and vertical semi-axis *b* in Bitmap *bp* with Code *f*.

The *eldisc* function draws an elliptical disc centered at *p* with horizontal semi-axis *a* and vertical semi-axis *b* in Bitmap *bp* with Code *f*.

The *eldiscture* function draws an elliptical disc centered at *p* with horizontal semi-axis *a* and vertical semi-axis *b* in Bitmap *bp* using Texture16 *t* with Code *f*.

The *elarc* function draws the corresponding elliptical arc, traveling counter-clockwise from the ellipse point closest to *p1* to the point closest to *p2*. Note: Differences exist between the calling conventions for *arc* and *elarc*.

EXAMPLE

The following routine can be used to allow a user to sweep out an ellipse by holding button 1 down. When button 1 is released, the ellipse is filled using the elliptical disc routine.

```
#include <dmd.h>

main()
{
        sweep_eldisc();
        request(KBD);
        wait(KBD);
}

sweep_eldisc()
{
    Point p, c;
    int a, b;

    request(MOUSE);
```

```
        while (!button1() )
           wait (MOUSE);
        c = p = mouse.xy;
        while (button1())
           if (!eqpt(p, mouse.xy)){
              if( !eqpt( p, c))  /* undraw old ellipse */
                 ellipse (&display, c, a, b, F_XOR);
              p = mouse.xy;
              a = abs (p.x - c.x);
              b = abs (p.y - c.y);
              ellipse (&display, c, a, b, F_XOR);
           }
        ellipse (&display, c, a, b, F_XOR);
        eldisc  (&display, c, a, b, F_XOR);
     }
```

SEE ALSO

circle(3L), jcircle(3L), jellipse(3L).

NAME

eq: eqpt, eqrect – compare for equality

SYNOPSIS

#include <dmd.h>

int eqpt (p, q)
Point p, q;

int eqrect (r, s)
Rectangle r, s;

DESCRIPTION

The *eqpt* function compares two Points and returns a 1 if the Points are equal or a 0 if they are unequal. Two Points are equal if the corresponding coordinates *x* and *y* are equal.

The *eqrect* function compares two Rectangles and returns a 1 if the Rectangles are equal or a 0 if they are unequal. Two Rectangles are equal if all four corresponding coordinates are equal.

EXAMPLE

The *eqmouse* function determines if the current mouse coordinate equals *p*. The *eqDrect* function determines if the Rectangle passed equals *Drect*.

```
#include <dmd.h>

eqmouse(p)
Point p;
{
        return eqpt (mouse.xy, p);
}

eqDrect(r)
Rectangle r;
{
        return eqrect (Drect, r);
}
```

SEE ALSO

structures(3R).

NAME

exit – cease execution

SYNOPSIS

void exit ( );

DESCRIPTION

The *exit* function terminates a process. Calling *exit* replaces the running process with the default terminal program. When a process calls *exit*, all local resources [keyboard, mouse, storage, etc.] are deallocated automatically. Any associated UNIX system process must be terminated separately.

Exit is called automatically when an application program returns from main( ).

When dmdio.h is included, *exit* acts differently. In addition to the above, it will terminate the host side of *jx*. Therefore, exit must be called explicitly by a process downloaded with *jx* to terminate the host side.

EXAMPLE

The following code fragment shows how a process can exit when a "q" is typed.

```
#include <dmd.h>

main()
{
        char c;

        request (KBD);

        .
        .

        if ( (c = kbdchar()) == 'q')
                exit();
        .
        .

}
```

NAME

fontname – get the name of a font

SYNOPSIS

#include <dmd.h>
#include <font.h>

char *fontname (f)

Font *f;

DESCRIPTION

The *fontname* function returns a pointer to the name of the font in the font cache that *f* points to. If **f* is not in the font cache, *fontname* returns a null character pointer.

SEE ALSO

fontrequest(3R), fontsave(3L), infont(3R/3L), structures(3R).

NAME

fontrequest, fontrelease, fontavail – request/release use of a font

SYNOPSIS

```
#include <dmd.h>
#include <font.h>

Font *fontrequest (fname)

void fontrelease (fname)

Font *fontavail (fname)

char *fname;
```

DESCRIPTION

Fname points to a font name, a null terminated string of up to 14 characters.

The *fontrequest* function returns a pointer to a font of the given name in the font cache. This routine will return 0 if one of the following conditions is true:

- there is no font of the given name in the cache,
- there is no more memory to attach the request information to the font,
- the calling process has already requested the named font.

While the font is requested, no process can remove it from the cache until it is released.

The *fontrelease* function tells the cache that the named font is no longer being used by the calling process. This is automatically done for all the fonts that a process has requested when that process exits or is deleted.

The *fontavail* function returns a pointer to the named font if it is in the font cache; 0, otherwise. This function is used only to check if a given font is in the cache, and it cannot substitute for *fontrequest* if the application intends to make use of the font.

SEE ALSO

fontname(3R), fontsave(3L), infont(3R/3L), structures(3R).

NAME

fontsave, fontcache, fontremove – save/remove a font from the cache

SYNOPSIS

```
#include <dmd.h>
#include <font.h>

int fontcache (fname, f)

Font *fontsave (fname, f)

void fontremove (fname)

char *fname;
Font *f;
```

DESCRIPTION

Fname points to a font name, a null terminated string of up to 14 characters.

The *fontsave* and *fontcache* functions put the given font into the font cache and give it the name *fname*. Once the font is in the cache, any other process within the terminal can use it by calling *fontrequest*.

The *fontcache* function expects the given font to be already in allocated memory through calls to *alloc* and *gcalloc*. This function also does a *fontrequest* for the calling application. If the caching or subsequent request fails, *fontcache* will return a 0; otherwise, a 1 is returned on success. Note that once the font is put into the cache, the font should only be freed by calling *fontremove*.

The *fontsave* function is used to cache fonts not in memory allocated through calls to *alloc* and *gcalloc*. *Fontsave* first attempts to allocate memory, then duplicates the given font into the allocated memory, and finally calls *fontcache* to cache the newly created font. On success, *fontsave* returns the pointer to the new font. If it cannot allocate enough memory for the creation of the new font, of if *fontcache* fails, *fontsave* will return a null pointer.

The *fontremove* function removes the named font from the cache and frees its memory. A request to remove a font that is currently requested by some other process will be ignored.

SEE ALSO

alloc(3R), fontname(3R), fontrequest(3R), gcalloc(3R), infont(3R/3L), structures(3R).

NAME

fontused, fontiname – font menu generator routines

SYNOPSIS

int fontused (fname)

char *fontiname (i)

char *fname;
int i;

DESCRIPTION

The *fontused* function tests if a given font has been requested by some process. This is a way to test if a call to *fontremove* will succeed. A font requested by a process cannot be removed. *Fontused* returns 1 if the font has been requested and 0 otherwise.

The *fontiname* function returns the name of the *ith+1* font in the font queue. This is useful for generating a menu of fonts in the cache. If there are less than *i+1* fonts, (char *)0 is returned.

EXAMPLE

The following example is a menu generator function that holds all the fonts in the cache. The fonts in use are greyed.

```
Titem *
fontmenu(i, m)
int i;
Tmenu *m;
{
   static Titem ti;
   int fontused();
   char *fontiname();

   if(ti.text = fontiname(i))
      ti.ufield.grey = fontused(ti.text);
   return(&ti);
}
```

SEE ALSO

fontname(3R), fontrequest(3R), fontsave(3L), tmenuhit(3R).

NAME

fpt: fPt, fRpt, fRect – create a Point or Rectangle from arguments

SYNOPSIS

```
#include <dmd.h>

Point fPt (x, y)
int x, y;

Rectangle fRpt (p, q)
Point p, q;

Rectangle fRect (a, b, c, d)
int a, b, c, d;
```

DESCRIPTION

The *fPt* function returns a point made from the two arguments.

The *fRpt* function returns a rectangle made from the two points.

The *fRect* function returns a rectangle made from the four arguments. This function differs from *canon(3R)* in that the points are not sorted first to guarantee a positive area.

The above functions are not macros as are the ones in *pt(3L)*; therefore, allow C language assignment constructs.

EXAMPLE

The following subroutine draws two boxes in the upper left corner of the window.

```
#include <dmd.h>

Point fPt();
Rectangle fRpt();
Rectangle fRect();

Point add();
Rectangle raddp();

drawboxes()
{
        Rectangle r;

        r = fRpt(Drect.origin, add(Drect.origin,
        Pt(100,100))); box(&display, r, F_STORE);

        r = fRect(0,0,200,200);
        box(&display, raddp(r, Drect.origin), F_STORE);
}
```

SEE ALSO

canon(3R), pt(3L).

NAME

frexp, ldexp, modf – manipulate parts of floating-point numbers

SYNOPSIS

double frexp (value, eptr)
double value;
int *eptr;

double ldexp (value, exp)
double value;
int exp;

double modf (value, iptr)
double value, *iptr;

DESCRIPTION

Every non-zero number can be written uniquely as $x * 2^n$, where the "mantissa" (fraction) x is in the range $0.5 \leq |x| < 1.0$, and the "exponent" n is an integer. *frexp* returns the mantissa of a double *value*, and stores the exponent indirectly in the location pointed to by *eptr*. If *value* is zero, both results returned by *frexp* are zero.

Ldexp returns the quantity $value * 2^{exp}$.

Modf returns the signed fractional part of *value* and stores the integral part indirectly in the location pointed to by *iptr*.

DIAGNOSTICS

If *ldexp* would cause overflow, ±**HUGE** (defined in **<ccs/math.h>**) is returned (according to the sign of *value*), and *errno* is set to **ERANGE**.
If *ldexp* would cause underflow, zero is returned and *errno* is set to **ERANGE**.

NAME

gcalloc, gcfree, gcallocown – garbage compacting memory allocation

SYNOPSIS

```
#include <dmd.h>

char *gcalloc (nbytes, where)

void gcfree (s)

void gcallocown (s, p)

unsigned long nbytes;
long **where;
char *s;
Proc *p;
```

DESCRIPTION

The *gcalloc* function provides a simple garbage-compacting memory allocator. It returns either a pointer to a block of *nbytes* contiguous bytes of storage or a NULL if unavailable. The storage is not initialized to zeros. The pointer *where* points to the location where the address of the storage block is to be saved. The contents of *where* will be updated after each compaction. That is, if garbage collection occurs, your storage will be moved, and your pointer will be changed to point to the new location of your storage area. Therefore, a program using *gcalloc* should never store the location of gcallocated memory anywhere other than the location handed to the allocator. Typically, this location is contained in a structure such as a *Bitmap*.

The *gcallocown* function changes the owning process of the memory returned by *gcalloc*. If *p* is zero, the memory will belong to no one and will only be freed by an explicit call to *gcfree*.

The *gcfree* function frees the storage block at *s*, thus making it available for future allocation.

The terminal automatically frees all memory *gcalloc*'ed and owned by a process when the process terminates or when the window it is running in is deleted. However, it is recommended that a process free its garbage compactable memory when the storage is no longer needed so that other processes will be able to use it.

EXAMPLE

These routines could be used for allocating and freeing space used to store Points in a polygon.

```
#include <dmd.h>

Point *poly;

Point *
polyalloc(n)
{
        char *gcalloc();
```

```
return (poly = (Point *)gcalloc(sizeof(Point)*n,
&poly));
}

polyfree( )
{
     void gcfree();

     gcfree (poly);
}
```

SEE ALSO

alloc(3R), balloc(3R), structures(3R).

DIAGNOSTICS

When garbage compaction is in effect, a small rectangle flashes on the upper left corner of the terminal's screen.

BUGS

If *gcalloc* is called to attempt to allocate over 7000000 bytes, it will fail, but it may say it succeeded. If this happens, the memory used by *gcalloc* is corrupted and may damage other programs running in the terminal.

NAME

getwbuf, putwbuf, Wbufsize – access the 630 MTG default terminal emulator buffer

SYNOPSIS

int getwbuf (buf, size)

int putwbuf (buf, size)

char *buf;
int size;

int Wbufsize;

DESCRIPTION

These functions give access to the global text buffer used by the 630 MTG default terminal emulator to hold the last cut, sent or saved text. They offer the ability for cut-and-paste communication between the default terminal emulator and user's applications.

The *getwbuf* function reads from the global text buffer. Up to *size* bytes are copied into the buffer pointed to by *buf*. The actual number of bytes copied is returned.

The *putwbuf* function writes from the character buffer pointed to by *buf* into the global text buffer. The number of bytes copied is returned. The previous content of the global text buffer is thrown away.

The global variable *Wbufsize* is the current number of characters in the global buffer used by *getwbuf* and *putwbuf*.

EXAMPLE

The following example is a function that takes the global text buffer and converts every 8 continuous spaces into a tab.

```
tabexpand()
{
   char buf1[120];
   char buf2[120];
   int i,j,k,space;

   k = 0;
   space = 0;
   i = getwbuf(buf1, 120);
   for(j=0; j<i; ++j)
   {
      buf2[k] = buf1[j];
      if(buf2[k] == ' ')
         space++;
      else
         space = 0;
      if(space == 8)
      {
```

```
            k -= 7;
            buf2[k] = '\t';
            space = 0;
        }
        ++k;
    }
    putwbuf(buf2, k);
}
```

NAME

globals: physical, display, Drect, Jrect, PtCurrent, P, mouse – globals describing display and mouse

SYNOPSIS

```
#include <dmd.h>

Bitmap physical;

Bitmap display;

Rectangle Drect;

Rectangle Jrect;

Point PtCurrent;

#define XMAX 1024
#define YMAX 1024

#define INSET 4

struct Mouse {
    Point xy;
    Point jxy;
    short buttons;
} mouse;

Proc *P;
```

DESCRIPTION

Each global is defined when **dmd.h** is included. One should not include these definitions in their source code.

The global *physical* is a Bitmap describing the entire screen display in screen coordinates.

The global *display* is the Bitmap describing an individual window in screen coordinates.

The global *Drect* is a Rectangle defining, in screen coordinates, the display area available to the program. It is not *display.rect,* which includes the border around each window.

The global *Jrect* is the Rectangle { 0, 0, *XMAX, YMAX* }

The global *PtCurrent* is the current Point in window coordinates which the j-routines reference and update.

The values *XMAX* and *YMAX* define the maximum x and y coordinates of the 630 MTG screen. These are the same as *physical.rect.corner.x* and *physical.rect.corner.y*.

The value *INSET* is the width of the border around a window. Therefore, *Drect* is the same as *inset(display.rect,INSET)*.

The global *mouse* is a location containing the current mouse coordinates and button states. The Point *xy* is in screen coordinates; *jxy* is in window coordinates. The *buttons* field is a bit vector of the mouse buttons that is most easily interpreted by using the button macros. The *mouse* is updated only

when requested and the window is current.

P is a special variable used by the 630 MTG. It represents the running process. A process can be a downloaded application or the default terminal emulator or an internal maintenance process. An example is the "control" process that puts up the button 3 global menu.

The 630 MTG changes the value of *P* whenever one process switches out through "wait" or "sleep" to let another run. It always points to the running process's table (Proc). Many routines either update *P* or use *P* to find information specific to the process. For example, the memory allocator "alloc" uses *P* to record who owns the memory requested. The "allocown" routine can be used to change this.

SEE ALSO

alloc(3R), btoc(3R), buttons(3R/3L), inset(3R), resources(3R), sleep(3R), structures(3R), transform(3R/3L).

WARNING

These globals (except XMAX, YMAX, and INSET) reside in the terminal. They cannot be used in automatic initializations of a program's global variables.

NAME

infont, getfont, outfont, ffree – read a font from the UNIX Operating system

SYNOPSIS

```
#include <dmd.h>
#include <font.h>

Font *infont (inch)
int (*inch)( );

Font *getfont (file)
char *file;

int outfont (f, ouch)
Font *f;
int (*ouch)( );

void ffree (f)
Font *f;
```

DESCRIPTION

The *infont* function creates a Font by reading the byte-wise binary representation returned by successive calls to *inch*. *Inch* must return successive bytes of the UNIX system file representation of the font, and -1 at end-of-file or on encountering an error. *Infont* will return 0 if memory to store the font cannot be allocated, or if the *inch* routine returns an error.

The *getfont* function is a higher level form of *infont* which can be used by programs running under *jx*. *Getfont* returns a pointer to a Font read from the named UNIX *file*. It accomplishes this by opening *file*, and then calling *infont* with the routine *getc* as an argument. The *getfont* function also returns 0 on error.

The *outfont* function calls the routine *ouch* to write successive bytes of the binary representation of Font **f*. The *outfont* function returns a –1 on error, as must the routine *ouch*.

Programs which use *infont* or *getfont* will normally want to cache the font with *fontcache*.

The *ffree* function frees the memory used by a Font allocated by *infont* or *getfont* if that font has not been added to the cache with *fontcache*. Programs which add fonts to the font cache with *fontcache* should release the font with *fontrelease*.

EXAMPLE

See the example for *string*(3R).

SEE ALSO

jx(1), fontrequest(3R), fontsave(3L), structures(3R).

NAME

inset – inset a border for a Rectangle

SYNOPSIS

#include <dmd.h>

Rectangle inset (r, n)
Rectangle r;
int n;

DESCRIPTION

The *inset* function returns the Rectangle:

{ *r.origin.x+n, r.origin.y+n, r.corner.x-n, r.corner.y-n* } .

EXAMPLE

The following simple program creates a clear Rectangle *r* with a 5-dot wide border inside *r*:

```
#include <dmd.h>

Rectangle inset();
Point add();

main()
{
        make_border();
        request(KBD);
        wait(KBD);
}

make_border()
{
        Rectangle r;
        Point s;
        s.x = 100;
        s.y = 100;

        r.origin = add(Drect.origin, s);
        r.corner = add(r.origin, s);
        rectf(&display, r, F_STORE);
        rectf(&display, inset(r, 5), F_CLR);
        return;
}
```

SEE ALSO

structures(3R).

NAME

integer: Iceil, Ifloor, min, max – integer functions

SYNOPSIS

short Iceil (a, b)

short Ifloor (a, b)

int min (b, c)

int max (b, c)

long a;
int b, c;

DESCRIPTION

The *Iceil* function returns the smallest short integer which, when multiplied by *b*, is not less than *a*.

The *Ifloor* function returns the largest short integer which, when multiplied by *b*, is not greater than *a*.

For both *Iceil* and *Ifloor* if *b* is equal to 0 or 1, the long integer *a* is truncated to a short and returned. Otherwise, if the result is greater than a short, it is truncated before being returned.

The *min* function returns the minimum of the two integers.

The *max* function returns the maximum of the two integers.

EXAMPLE

```
Iceil (7, 2)  and  Iceil (10, 3) both equal 4.

Ifloor (9, 2)  and  Ifloor (13, 3) both equal 4.

min (32, 16) equals 16.

max (32, 14) equals 32.
```

NAME

ismpx – test if connected to a multiplexed host

SYNOPSIS

int ismpx ( );

DESCRIPTION

The *ismpx* function tests if the application program is connected to a host via a multiplexed connection. If so, a 1 is returned; otherwise, 0 is returned. *Layers* is one example of a multiplexed connection protocol.

SEE ALSO

attach(3R), local(3R), peel(3R).
layers(1) in *UNIX System V Release 3 User's Reference Manual.*
layers(1) in *5620 Dot-Mapped Display Reference Manual.*

NAME

itox, itoa, itoo – convert integer to string representation

SYNOPSIS

char *itox (n, s)

char *itoa (n, s)

char *itoo (n, s)

long n;
char *s;

DESCRIPTION

The *itox* function returns the hexadecimal string representation of the long integer *n* prefixed with "0x".

The *itoa* function returns the decimal string representaion of the integer *n*.

The *itoo* function returns the octal string representation of the integer *n* prepended with "0".

The argument *s* must point to a buffer large enough to hold the string. The return value is *s*.

EXAMPLE

The following example produces a hexadecimal string without the "0x" prefix.

```
#include <dmd.h>
char *itox();

char *
myitox(buf)
char *buf;
{
        return(itox((long)192, buf) + 2);
}
```

Note: the return value is *buf* + 2, which is the hexadecimal string "C0".

NAME

itrig: Icos, Isin, Iatan2 – cosine, sine and arc tangent trigonometric functions

SYNOPSIS

int Icos (d)

int Isin (d)

int Iatan2 (x, y)

int d;
int x, y;

DESCRIPTION

The *Icos* and *Isin* functions return scaled integer approximations to the trigonometric functions. The argument values are in degrees. The values returned are scaled so that **Icos(0)==1024**.

The *Iatan2* function returns the approximate arc-tangent of y/x. The return value is in integral degrees. The error in approximation may be as large as five degrees.

EXAMPLE

These routines can be used to calculate mathematical expressions such as:

```
x=x0*Icos(d)
```

or to calculate a projection:

```
x=muldiv(x0, Icos(d), 1024)
```

Note, the multiplication must be scaled.

SEE ALSO

muldiv(3L).

NAME

jcircle, jdisc, jarc – draw circle on display

SYNOPSIS

#include <dmd.h>

void jcircle (p, r, f)

void jdisc (p, r, f)

void jarc (p, p1, p2, f)

Point p, p1, p2;
int r;
Code f;

DESCRIPTION

The *jcircle* function draws the approximate circle of radius *r* centered at *p* with Code *f* in the display bitmap.

The *jdisc* function draws a disc of radius *r* centered at *p* with Code *f* in the display bitmap.

The *jarc* function draws the circular arc centered at *p* counterclockwise from *p1* to the point on the circle closest to *p2* with Code *f* in the display bitmap.

All coordinates and radii are in window coordinates. Because the window is scaled, these routines are actually implemented by calls to the ellipse routines.

EXAMPLE

The following routine draws a row of eight circles, a row of eight discs, and a row of eight arcs, scaled to the shape of the window.

```
#include <dmd.h>

draw()
{
  Point p;
  int i, r;

  r = 50;
  p.y = 200;
  for (p.x=100; p.x < XMAX-50; p.x+=120)
       jcircle (p, r, F_XOR);
  p.y = 600;
  for (p.x=100; p.x < XMAX-50; p.x+=120)
       jdisc (p, r, F_XOR);
  p.y = 900;
  for (p.x=100; p.x < XMAX-50; p.x+=120)
       jarc (p, Pt(p.x-r, p.y), Pt(p.x+r,
            p.y), F_XOR);

  request(KBD);
  wait(KBD);
}
```

SEE ALSO
circle(3L), ellipse(3L), globals(3R), jellipse(3L), structures(3R), transform(3R/3L).

NAME

jellipse, jeldisc, jelarc – draw ellipse on display

SYNOPSIS

#include <dmd.h>

void jellipse (p, a, b, f)

void jeldisc (p, a, b, f)

void jelarc (p, a, b, p1, p2, f)

Point p, p1, p2;
int a, b;
Code f;

DESCRIPTION

The *jellipse* function draws an approximate ellipse centered at *p*, with horizontal semi-axis *a* and vertical semi-axis *b* with Code *f* in the display bitmap.

The *jeldisc* function draws an elliptical disc centered at *p*, with horizontal semi-axis *a* and vertical semi-axis *b* with Code *f* in the display bitmap.

The *jelarc* function draws the corresponding elliptical arc, centered at *p*, counterclockwise from the ellipse point closest to *p1* to the ellipse point closest to *p2* with Code *f* in the display bitmap.

All coordinates and semi-axes are in window coordinates.

EXAMPLE

The following routine draws a row of eight ellipses, a row of eight discs, and a row of eight arcs, scaled to the shape of the window.

```
#include <dmd.h>

draw()
{
  Point p;
  int i, r;

  r = 50;
  p.y = 200;
  for ( p.x=100; p.x < XMAX-50; p.x+=120)
       jellipse (p, r, r-25, F_XOR);
  p.y = 600;
  for ( p.x=100; p.x < XMAX-50; p.x+=120)
       jeldisc (p, r-25, r, F_XOR);
  p.y = 900;
  for (p.x=100; p.x < XMAX-50; p.x+=120)
       jelarc (p, r, r, Pt(p.x-r, p.y),
                Pt(p.x+r, p.y), F_XOR);

  request(KBD);
  wait(KBD);
}
```

SEE ALSO

circle(3L), ellipse(3L), globals(3R), jcircle(3L), structures(3R), transform(3R/3L).

NAME

jmove, jmoveto – move current window point on display, relative or absolute

SYNOPSIS

#include <dmd.h>

void jmove (p)

void jmoveto (p)

Point p;

DESCRIPTION

The *jmove* function moves the current window point by the relative vector *p* which is in window coordinates.

The *jmoveto* function sets the current window point to the absolute location *p* which is in window coordinates.

EXAMPLE

See the example in *jsegment*(3L).

SEE ALSO

jsegment(3L), moveto(3L), structures(3R), transform(3R/3L).

NAME

jpoint - draw single pixel on display

SYNOPSIS

```
#include <dmd.h>

void jpoint (p, f)
Point p;
Code f;
```

DESCRIPTION

The *jpoint* function sets the pixel at location *p* (in window coordinates) in the display bitmap according to the Code *f*.

EXAMPLE

The following routine can be used to "doodle" on the screen.

```
#include <dmd.h>

doodle()
{
        request (MOUSE);
        for ( ;; ) {
            wait (MOUSE)
            if( button3() )
                break;
            if( button2() )
                jpoint (mouse.jxy, F_STORE);
        }
}
```

SEE ALSO

globals(3R), point(3R), structures(3R), transform(3R/3L).

NAME

jrectf – rectangle function on display

SYNOPSIS

#include <dmd.h>

void jrectf (r, f)
Rectangle r;
Code f;

DESCRIPTION

The *jrectf* function performs the action specified by the Code *f* on the Rectangle *r* in the display bitmap. The Rectangle *r* is in window coordinates.

EXAMPLE

The following subroutine will "doodle" on the screen using a Rectangle, whose coordinates are scaled to the window.

```
#include <dmd.h>

Point add();

rectdoodle()
{
     Rectangle r;
     Point s;

     s.x = 16;
     s.y = 16;
     request (MOUSE);
     for (;;) {
          wait(MOUSE)
          r.origin = mouse.jxy;
          r.corner = add (r.origin, s);
          if ( button3() )
             break;
          if ( button1() )
             jrectf (r, F_STORE);
          if ( button2() )
             jrectf (r, F_CLR);
     }
}
```

SEE ALSO

globals(3R), rectf(3R), structures(3R), transform(3R/3L).

NAME

jsegment, jline, jlineto – draw line on display

SYNOPSIS

```
#include <dmd.h>

void jsegment (p, q, f)
void jline (p, f)
void jlineto (p, f)
Point p, q;
Code f;
```

DESCRIPTION

The *jline* function draws a line in the display bitmap with Code *f* from the current window point (initially (0, 0) in window coordinates) along the relative vector *p* which is in window coordinates.

The *jlineto* function draws a line in the display bitmap from the current window point to the absolute window coordinate *p* with Code *f*.

The *jsegment* function draws a line in the diplay bitmap from the window coordinate *p* to the window coordinate *q* with Code *f*.

The line functions *jline*, *jlineto*, and *jsegment* leave the current window point at the end of the line.

PtCurrent is the global used to refer to the current window point.

EXAMPLE

The following program draws three boxes on the screen using three different methods.

```
#include <dmd.h>

main()
{
    box(Rect(400,100,600,300));
    rbox(Rect(0,0,200,200),Pt(0,300));
    sbox(Rect(400,700,600,900));
    request(KBD);
    wait(KBD);
}

/* draw absolute */
box(r)
Rectangle r;
{
    jmoveto (r.origin);
    jlineto (Pt (r.corner.x, r.origin.y), F_XOR);
    jlineto (r.corner, F_XOR);
    jlineto (Pt (r.origin.x, r.corner.y), F_XOR);
    jlineto (r.origin, F_XOR);
}
```

```
/* draw relative */
rbox(r, p)
Rectangle r;
Point p;
{
   jmove (p);
   jline (Pt (r.corner.x - r.origin.x, 0), F_XOR);
   jline (Pt (0, r.corner.y - r.origin.y), F_XOR);
   jline (Pt (r.origin.x - r.corner.x, 0), F_XOR);
   jline (Pt (0, r.origin.y - r.corner.y), F_XOR);
}

/* draw with segments */
sbox(r)
Rectangle r;
{
   jsegment (r.origin, Pt(r.corner.x,
             r.origin.y), F_XOR);
   jsegment (Pt(r.corner.x, r.origin.y),
             r.corner, F_XOR);
   jsegment (r.corner, Pt(r.origin.x,
             r.corner.y), F_XOR);
   jsegment (Pt(r.origin.x, r.corner.y),
             r.origin, F_XOR);
}
```

SEE ALSO

globals(3R), pt(3L), segment(3R), structures(3R), transform(3R/3L).

NAME

jstring – draw character string on display

SYNOPSIS

#include <dmd.h>

Point jstring (s)
char *s;

DESCRIPTION

The *jstring* function draws, in the F_XOR mode, the null-terminated string *s* in the display Bitmap using mediumfont so that the origin of the rectangle enclosing the first character is at the current window point. It returns the current window point, which is left after the last character of the string, so adjacent calls to *jstring* can appear to concatenate their argument strings on the screen.

EXAMPLE

The following program shows how *jstring* automatically updates *PtCurrent*.

```
#include <dmd.h>

Point jstring();

main()
{
    jmove (Pt(16,16));
    jstring ("I");
    jline (Pt(40,0), F_XOR);
    jstring ("love");
    jline (Pt(40,0), F_XOR);
    jstring ("my");
    jline (Pt(40,0), F_XOR);
    jstring ("630MTG");

    request (KBD);
    wait (KBD);
}
```

SEE ALSO

globals(3R), string(3R), structures(3R).

NAME

jtexture – draw Texture in Rectangle on display

SYNOPSIS

#include <dmd.h>

void jtexture (r, t, f)
Rectangle r;
Texture16 *t;
Code f;

Texture16 T_grey, T_lightgrey, T_darkgrey;

Texture16 T_black, T_white, T_background, T_checks;

DESCRIPTION

The *jtexture* function fills the Rectangle *r* in the display Bitmap with Texture16 *t* using function Code *f*. The Rectangle *r* is in window coordinates. The Texture16s listed above are predefined.

EXAMPLE

The following routine allows one to doodle with a Texture16.

```
#include <dmd.h>

Point add();

main()
{
    Rectangle r;
    Point s;

    s.x = 16;
    s.y = 16;
    request (MOUSE);
    for (;;) {
        wait(MOUSE)
        r.origin = mouse.jxy;
        r.corner = add (r.origin, s);
        if ( button3() )
          break;
        if ( button1() )
          jtexture (r, &T_grey, F_STORE);
        if ( button2() )
          jtexture (r, &T_grey, F_CLR);
    }
}
```

SEE ALSO

globals(3R), structures(3R), texture(3R), transform(3R/3L).

NAME

kbdchar – read character from keyboard

SYNOPSIS

int kbdchar ( )

DESCRIPTION

The *kbdchar* function returns the next keyboard character typed to the process. If no characters have been typed, *kbdchar* returns –1. If *KBD* has not been requested, *kbdchar* will always return –1, even if characters have been typed to the process.

EXAMPLE

This code will prevent a program from exiting until the 'q' character has been typed on the keyboard.

```
#include <dmd.h>

main( )
{
     request (KBD);

     .
     .
     .

     do
          wait(KBD);
     while( kbdchar ( ) != 'q' );
}
```

SEE ALSO

resources(3R).

WARNING

Since the keyboard routine is a process, you must release the CPU in order to have typed characters placed on the application's keyboard queue.

NAME

keyboard: P->state, SCRLOCKREQD, SCR_LOCK, NOPFEXPAND, NOCURSEXPAND, NOPADEXPAND, NOTRANSLATE, reqkbdID( ) - per process keyboard states, keyboard ID

SYNOPSIS

#include <dmd.h>

long P->state;

void reqkbdID ( )

DESCRIPTION

P->state is the state variable for an application running in the 630 MTG.

An application can give different interpretations on groups of 630 MTG supported keyboard keys by setting corresponding bits in the process state variable. The groups are Scroll Lock key, programmable function keys, arrow keys, numerical keypad keys, or the whole keyboard. If the bits are not set, when a key is depressed, the keyboard will send to the process the ASCII/hexadecimal code(s) as specified by the *630 MTG Terminal User's Guide*. An exception is the Scroll Lock key which is a "dead" key (no special processing) if it is not requested and processed by the application itself.

If an application program wishes to implement local terminal flow control with the keyboard Scroll Lock key, it must request the use of the Scroll Lock key by setting the SCRLOCKREQD in the state variable as follows:

```
P->state |= SCRLOCKREQD;
```

The application program can then determine if the Scroll Lock key has been depressed by checking the SCR_LOCK bit as follows:

```
P->state & SCR_LOCK;
```

The SCR_LOCK bit is set whenever the Scroll Lock key is depressed. It is automatically cleared by the keyboard when the Scroll Lock key is depressed a second time.

The programmable function (PF) keys, when depressed are expanded by default, i.e., the corresponding character strings (minus the NULL character) stored in the non-volatile BRAM memory will be sent to the process as would be typed from the keyboard. If an application prefers to have its own interpretation of the programmable function keys, it is necessary to set a bit in the process state variable:

```
P->state |= NOPFEXPAND;
```

In this case, the keyboard will only send to the process the PF key keycode which is 0x80 for PF key F1, 0x81 for PF key F2, and so forth.

If a program wants to interpret the arrow keys without having to parse their entire sequence, it can set a bit in the process state variable:

```
P->state |= NOCURSEXPAND;
```

When an arrow key is then depressed, it will have the value:

up arrow	0xE0
down arrow	0xE1
right arrow	0xE2
left arrow	0xE3
home key	0xE4

If a program wants to have its own interpretation of the numerical keypad keys, it is necessary to set a bit in the process state variable:

```
P->state |= NOPADEXPAND;
```

When a numerical keypad key is then depressed, it will have the value:

Enter	0xC0
=	0xC1
*	0xC2
/	0xC3
+	0xC4
7	0xC5
8	0xC6
9	0xC7
-	0xC8
4	0xC9
5	0xCA
6	0xCB
,	0xCC
1	0xCD
2	0xCE
3	0xCF
0	0xD0
.	0xD1

If a program wants to remap the entire keyboard, it must set a bit in the state variable:

```
P->state |= NOTRANSLATE;
```

The application will then receive from the keyboard the raw keycodes as listed in the *630 MTG Terminal User's Guide*. Also, the keyboard LEDs will no longer be automatically updated. At this point, calling the *reqkbdID* function will ask the keyboard for its identity (ID). The keyboard will send back its ID and all keys currently depressed, followed again by the ID. Refer to the *630 MTG Software Development Guide* for more information.

SEE ALSO

kbdchar(3R), state(3R), pfkey(3R), resources(3R), setled(3L).
630 MTG Terminal User's Guide.
630 MTG Software Development Guide.

WARNING

If the process has not requested the keyboard (see *resources(3r)*), all per process keyboard states will be ignored, and the key's ASCII/hexadecimal code(s) as specified in the *630 MTG Terminal User's Guide* will be sent to the host (if the process is connected) or ignored (if the process runs locally).

The exception is the Scroll Lock key which does not require the application to request the whole keyboard. However in order to process that key, the application has to set the SCRLOCKREQD bit in *P->state*.

NAME

labelon, labeloff, labelicon, labeltext – window labeling

SYNOPSIS

```
#include <dmd.h>
#include <label.h>

void labelon ( )

void labeloff ( )

void labelicon (bp, pos)
Bitmap *bp;
int pos;

labeltext (s, n, f)
char *s;
int n;
int f;
```

DESCRIPTION

The *labelon* function puts a label at the top of the window. The label area is **LABEL_HEIGHT** pixels high and spans the full interior width of the window. The label area is automatically updated to indicate the window's current host and if that host is multiplexed. It also indicates if scroll lock or caps lock is active for that window and if the application running in the window has requested the printer. **Drect** is changed to the new smaller size of the interior of the window.

The *labeloff* function removes the label from the window and changes **Drect** to the new larger size.

The *labelicon* function draws the given bitmap into the label. The bitmap is clipped to 16 pixels high and to the right edge of the label area. Its leftmost edge is aligned to position *pos*. Positions are 16 pixels apart, starting with position 1 (L_HOST_POSITION) at the left edge of the label area. The first 5 positions (1 to 5) are used by the terminal for the following default information:

L_HOST_POSITION	current host connection
L_MUX_POSITION	current host environment
L_PRINT_POSITON	printer request status
L_SCROLL_POSITION	scroll lock key status
L_CAP_POSITION	caps lock key status

It is not recommended for an application to override those fields with its own bitmaps. The first position index that an application should use is **L_USER_POSITION** which guarantees non-interference with the terminal.

Clipped portions of a label bitmap are not remembered by the terminal if the window is made larger.

The function *labeltext* displays the string *s* of *n* characters in the label area of the window based on the flag *f*. The possible values for f are:

L_LEFT	left justify the string from the L_USER_POSITION index position
L_RIGHT	right justify the string in the label area
L_CENTER	center the string in the full length label area

The font used to display the string is the terminal's **medium** font. The display mode is F_XOR; therefore, *labeltext* will superimpose the new string over any existing label strings. If there is not enough room in the label area for all characters to fit, the string will be clipped off.

When the window is reshaped, the terminal only redraws the label area with the default information. Any user supplied icons and strings must be redrawn by the application.

Any application that makes use of both *labelicon* and *labeltext* should make sure that they do not write to the same positions of the label area.

NAME

local – make the calling process local

SYNOPSIS

int local ( )

DESCRIPTION

The *local* function provides the application developer with the means to change the disposition of a process from the connected state (e.g. using a *layers* connection and host resources) to a local state. The host side processes are killed as if the process were deleted. However the terminal retains all local information and other resources that were requested through the *request(3R)* function. If the process does not own the KBD resource, typing on the keyboard will sound the bell.

When a process is made local, the border surrounding the process's window is changed to a checkered pattern to differentiate it with connected windows.

The *local* function will fail if the process is already local or is the last process connected to the host. Therefore, the *local* function will always fail if the process is operating in the non-layers environment which, by definition, supports only one host connection. A *local* function failure results in a zero value being returned. On success, a 1 is returned.

SEE ALSO

attach(3R), peel(3R).
layers(1) in *UNIX System V Release 3 User's Reference Manual.*
layers(1) in *5620 Dot-Mapped Display Reference Manual.*

NAME

lputchar – 630 MTG local putchar function

SYNOPSIS

void lputchar (c)
char c;

DESCRIPTION

Lputchar is syntactically equivalent to the UNIX standard I/O putchar function. It can be called by downloaded application programs who want to display characters within their window on the 630 MTG screen. Where (within the applications window) characters are displayed can be affected with the *moveto*(3L) function. *Lputchar* calls the *moveto* routine to update the *current screen point* after it displays the charcter *c*.

How characters are eventually displayed on a user's terminal when using the UNIX putchar function is affected by the UNIX host *stty*(1) settings and the user's terminal characteristics. Since *lputchar* displays directly onto the 630 MTG screen, assumptions were made about desired *stty* settings. In general, *lputchar* does as little processing on the output stream as practical. If more extensive processing of control characters is needed, it can be accomplished as shown in the example program below.

The following are the only control characters processed by *lputchar*. All other characters will be displayed as **ASCII** characters:

\r Carriage Return. Move the current point to the left edge of the window.

\n Newline. Move the current point down one line and to the left edge of the window. Scroll the window if necessary.

\t Horizontal tab. Tab characters are expanded to spaces with tab stops at every eighth space.

EXAMPLE

If an application program wanted to, for example, process bells and backspaces rather than display the **ASCII** bell and backspace characters, it could define a function *mylputchar* as follows:

```
#include <dmd.h>
#include <font.h>

void lputchar();
void moveto();
void ringbell();
Point sPtCurrent();

mylputchar(c)
char c;
{
        Point curpos;

        switch( c ) {
        case '\007':
```

```
                ringbell();
                break;
        case '\b':
                /* get the current position
                ** if( not already at the left edge of the
                **    window ) {
                **    move back one character position
                **    lputchar a space to erase the last
                **    character and move back again
                ** }
                */
                curpos = sPtCurrent();
                if(curpos.x - FONTWIDTH(largefont)
                     >= Drect.origin.x) {
                        curpos.x -= FONTWIDTH(largefont);
                        moveto(curpos);
                        lputchar(' ');
                        moveto(curpos);
                }
                break;
        default:
                lputchar(c);
                break;
        }
}
```

SEE ALSO

bputchar(3L), moveto(3L), printf(3L), structures(3R).
stty(1) in the *UNIX System V User's Reference Manual.*
putc(3S) in the *UNIX System V Programmer's Reference Manual.*

NAME

lsearch – linear search and update

SYNOPSIS

#include <ccs/search.h>

char *lsearch ((char *)key, (char *)base, nelp, sizeof(*key), compar)
unsigned *nelp;
int (*compar)( );

DESCRIPTION

Lsearch is a linear search routine generalized from Knuth (6.1) Algorithm S. It returns a pointer into a table indicating where a datum may be found. If the datum does not occur, it is added at the end of the table. **Key** points to the datum to be sought in the table. **Base** points to the first element in the table. **Nelp** points to an integer containing the current number of elements in the table. The integer is incremented if the datum is added to the table. **Compar** is the name of the comparison function which the user must supply (*strcmp*, for example). It is called with two arguments that point to the elements being compared. The function must return zero if the elements are equal and non-zero otherwise.

NOTES

The pointers to the key and the element at the base of the table should be of type pointer-to-element, and cast to type pointer-to-character.
The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.
Although declared as type pointer-to-character, the value returned should be cast into type pointer-to-element.

EXAMPLE

This fragment will read in less than TABSIZE strings of length less than ELSIZE and store them in a table, eliminating duplicates.

```
#include  <dmd.h>
#include  <dmdio.h>
#include  <ccs/search.h>

#define  TABSIZE  50
#define  ELSIZE  120

      char  line[ELSIZE],  tab[TABSIZE][ELSIZE],
                  *lsearch( );
      unsigned  nel  =  0;
      int  strcmp( );
      . . .
      while  (fgets(line,  ELSIZE,  stdin)  !=  NULL  &&
           nel  <  TABSIZE)
            (void)  lsearch(line,  (char  *)tab,  &nel,
                        ELSIZE,  strcmp);
      . . .
```

SEE ALSO

bsearch(3L), str(3L).

DIAGNOSTICS

If the searched for datum is found, *lsearch* return a pointer to it. Otherwise, *lsearch* returns a pointer to the newly added element.

BUGS

Undefined results can occur if there is not enough room in the table to add a new item.

NAME

lsqrt – integer square root

SYNOPSIS

long lsqrt (x)
long x;

DESCRIPTION

The *lsqrt* function returns the signed long integer closest to the square root of the signed long argument.

SEE ALSO

norm(3R).

NAME

memory: memccpy, memchr, memcmp, memcpy, memset – memory operations

SYNOPSIS

```
#include <ccs/memory.h>

char *memccpy (s1, s2, c, n)
char *s1, *s2;
int c, n;

char *memchr (s, c, n)
char *s;
int c, n;

int memcmp (s1, s2, n)
char *s1, *s2;
int n;

char *memcpy (s1, s2, n)
char *s1, *s2;
int n;

char *memset (s, c, n)
char *s;
int c, n;
```

DESCRIPTION

These functions operate as efficiently as possible on memory areas (arrays of characters bounded by a count, not terminated by a null character). They do not check for the overflow of any receiving memory area.

Memccpy copies characters from memory area **s2** into **s1**, stopping after the first occurrence of character **c** has been copied, or after **n** characters have been copied, whichever comes first. It returns a pointer to the character after the copy of **c** in **s1**, or a NULL pointer if **c** was not found in the first **n** characters of **s2**.

Memchr returns a pointer to the first occurrence of character **c** in the first **n** characters of memory area **s,** or a NULL pointer if **c** does not occur.

Memcmp compares its arguments, looking at the first **n** characters only, and returns an integer less than, equal to, or greater than 0, according as **s1** is lexicographically less than, equal to, or greater than **s2**.

Memcpy copies **n** characters from memory area **s2** to **s1**. It returns **s1**.

Memset sets the first **n** characters in memory area **s** to the value of character **c**. It returns **s**.

For user convenience, all these functions are declared in the optional <**ccs/memory.h**> header file.

WARNING

Memcmp is implemented by using the most natural character comparison on the machine. Thus the sign of the value returned when one of the characters has its high order bit set is not the same in all implementations and should not be relied upon.

Character movement is performed differently in different implementations. Thus overlapping moves may yield surprises.

NAME

menuhit – present user with menu and get selection

SYNOPSIS

#include <dmd.h>

int menuhit (m, n)

Menu *m;
int n;

```
typedef struct Menu {
   char   **item;           /* string array, ending with 0 */
   short  prevhit;          /* retained from previous call */
   short  prevtop;          /* retained from previous call */
   char   *(*generator)(); /* used if item == 0 */
} Menu;
```

DESCRIPTION

The *menuhit* function presents the user with a menu specified by the Menu pointer *m* and returns an integer indicating the selection made. A returned 0 would indicate that the first item in the menu had been selected; if a 1 is returned, the second item has been selected, etc. A -1 indicates no selection. The *n* argument is an integer which specifies which button to use for the interaction: 1, 2 or 3. The *menuhit* function assumes that the button is already depressed when it is called. The user makes a selection by lifting the button when the cursor points to the desired selection. Lifting the button outside the menu indicates no selection.

The maximum number of menu items displayed at any one time is 16 items. When the number of items is 16 or less, all the items are displayed and are centered one entry per line in the menu. This is the normal menu mode. When there are more than 16 menu items to be displayed, the menu becomes a **scrolling** menu. The left portion of the menu contains a scroll bar which is used for scrolling quickly through the menu selections. The vertical size of the scroll bar is an indication of the size of the user's view of the menu (16 items) relative to the number of selections in the entire menu.

There are two ways to scroll through the menu items. The first is to move the mouse cursor to the left side of the menu into the scroll bar area. By moving the mouse cursor up or down within the scroll bar area, the menu items will scroll accordingly. The second method used to scroll through the menu items is to place the mouse cursor on the top or bottom entry of the menu list. The menu will scroll up or down by one item at a time if there are additional items to be displayed in that direction.

The *prevhit* variable is used to store the menu's previous selection. When *menuhit* is called, the menu is displayed such that, if possible, the mouse cursor will be displayed over the previous selection. *Prevhit* holds the index from the top of the displayed menu. The *prevtop* variable is used to store the previous topmost item displayed in a scrolling menu. The values of *prevhit* and *prevtop* are initialized to 0 and need not normally be

manipulated by the application program.

Menus may be generated dynamically from a program by specifying a generator function in the *Menu* structure. If *item* is set to 0 when *menuhit* is called, then the routine specified by *generator* is called with one parameter which is an integer index beginning at 0. The generator must return a pointer to a character string containing the text for the corresponding menu item. This generator function is called repeatedly with the index increasing by 1 until the generator returns a NULL, indicating the end of the menu selections. The generator function is called each time the *menuhit* routine is called with item set to NULL (i.e., 0).

Another facility provided by *menuhit* is that of a spread character. A spread character is any ascii character with the high-order bit set. The spread character acts somewhat like a spring pushing against the adjacent text and borders within a menu entry. The spread character can be placed at the beginning, middle, or end of the string defining the menu entry. If placed at the beginning of the string, the text in the menu item will be right-justified. If placed at the end of the string, the text will be left-justified. If placed in the middle of the string, the text on each side of the spread character will be pushed against the corresponding menu border. In each case, the space created by the spread character will be filled in with the ascii character contained in the spread character. For entries without a spread character, the default is to have the text centered.

Whenever a menu is displayed, the original screen image obscured by the menu is saved in the terminal and then later restored when the menu disappears. If the terminal is out of memory and therefore cannot save the screen image, then the menu will be displayed in XOR (exclusive or) mode on top of the existing screen image. Menu items may still be selected in this mode but the items might be hard to read. To remedy this problem, memory may be freed up by either deleting or reshaping windows before the menu is displayed.

EXAMPLE

The following example includes both a menu with the spread character and a menu that is dynamically generated. Button 2 and Button 3 are used to bring up the two menus and Button 1 exits.

```
#include <dmd.h>

char *menutext[] = {
        "left\240",             /* space char with
                                high bit set */
        "\256right",            /* . char with high
                                   bit set */
        "middle",
        "left\337right",        /* _ char with high
                                   bit set */
        "a very long string",
        NULL };
Menu menu = { menutext };       /* static menu */
```

```
/*     Note the above menu will appear as:

        --------------------
        |left                |
        |.............right|
        |      middle        |
        |left _________right|
        |a very long string|
        --------------------
*/

char scrlstr[8]="scroll";

char *
generate(i)
int i;
{
    if (i>99)   /* generator stopping condition */
        return NULL;
    else {  /* generate test for items (ie. "scroll56") */
        scrlstr[6] = i/10 + '0';
        scrlstr[7] = i - (i/10 *10) + '0';
        scrlstr[8] = '\0';
    }
    return scrlstr;

}


Menu menugen = {0, 0, 0, generate};
            /* dynamically generated menu */


main()
{
       int m;

   for (;;) {
       request(MOUSE);
       wait(MOUSE);
       if (button1())
          break;
       else if (button2()) {
          m = menuhit(&menugen,2);
          lprintf("your selection was %d\n",m);
       }
       else if (button3()) {
          m = menuhit(&menu,3);
          lprintf("your selection was %d\n",m);
       }
```

```
        }
    }
```

SEE ALSO
tmenuhit(3R).

NAME

moveto, sPtCurrent – change and return the value current screen point

SYNOPSIS

#include <dmd.h>

void moveto (p)
Point p;

Point sPtCurrent ( )

extern int didmoveto;

DESCRIPTION

These functions can be used to change or return the value of the current screen point. The current screen point is simply a place holder that applications can use to manage current screen position. For example, the *lputchar* function uses *sPtCurrent* to find the point at which to print the next character and uses *moveto* to update the current screen point after it prints the character.

The current screen point is similar to but distinct from *PtCurrent* (see *globals*(3R)). The primary difference is that the current screen point is stored in screen coordinates, and the *PtCurrent* is stored in window coordinates (see *transform*(3R)). This makes the current screen point easier to deal with for applications that want to work completely in screen coordinates.

The *moveto* function will move the current screen point to the point *p*. The *sPtCurrent* function returns the value of the current screen point.

The current screen point is actually stored in the variable P->scurpt. It is stored as an offset from Drect.origin (i.e., sub[p, Drect.origin]). Note that this refers only to internal representation. The functions *moveto* and *sPtCurrent* work in actual screen coordinates and translate the offset on each call.

Storing the current screen point as an offset from Drect.origin has the advantage that successive calls to *sPtCurrent* will return the proper position within a window even if the window was moved between calls.

Reshapes of a window between successive calls to *sPtCurrent* are handled as follows. If the offset of the current screen point from Drect.origin is still within the window after the reshape, *sPtCurrent* will return the current screen point within the new window at the same offset from Drect.origin that existed in the old window. If the offset from Drect.origin is no longer within the window (i.e., the window was reshaped smaller), *sPtCurrent* will return Drect.origin as the current screen point. If an application wants to handle reshape more elegantly, it can use the following code fragment after each call to the *wait* function. This code fragment will cause the current screen point to move to the upper left-hand corner of the window after a reshape.

```
if(P->state&RESHAPED) {
        if(!(P->state&MOVED))
                moveto(Drect.origin);
        P->state &= ~(MOVED|RESHAPED);
}
```

The current screen point must be initialized with the *moveto* function before *sPtCurrent* is called the first time. Library routines which use this facility can check if initialization is necessary by looking at the global variable *didmoveto* each time they are called. This variable will be set to 0 if *moveto* has not been called. An example below shows how the *didmoveto* variable is used to determine if initialization is necessary within a simple putchar function.

EXAMPLES

There are two types of users of the current screen point. The first type of user is calling existent library routines such as *lprintf* and *lputchar*, and is only interested in using the *moveto* function to control the library routines. The following code fragment illustrates how *moveto* can be used with *lprintf* to display a prompt at the bottom of the window.

```
#include <dmd.h>
#include <font.h>

extern Point fPt();
Point p;

p = fPt( Drect.origin.x,
    Drect.corner.y - FONTHEIGHT(largefont) );
moveto(p);
lprintf("Choose an Option> ");
```

The second type of user of the current screen point is writing new library routines which use this facility. The following example shows how to accomplish this by implementing a simple putchar routine. In the example below, didmoveto is checked first to see if initialization of the current point is required. Then the code obtains the value of the current screen point, prints a character, and updates the current screen point for the next call to myputchar.

```
myputchar(c)
char c;
{
        extern int didmoveto;
        extern Point sPtCurrent();
        extern Point string();
        char s[2];
        Point curpos;

        s[0] = c;
        s[1] = '\0';

        if(!didmoveto)
                  moveto(Drect.origin);

        curpos = sPtCurrent();
        curpos = string(&largefont, s, &display,
                    curpos, F_STORE);
        moveto(curpos);

}
```

SEE ALSO

globals(3R), jmove(3R), printf(3L), lputchar(3L), resources(3R), structures(3R), transform(3R/3L).

BUGS

The **didmoveto** initialization scheme will not work with shared text applications because *didmoveto* is a global variable shared by all invocations of a shared text application. Shared text applications must explicitly initialize the current screen point by calling *moveto*.

NAME

msgbox – put up a message in a box

SYNOPSIS

int msgbox (s,...)
char *s;

DESCRIPTION

The *msgbox* function puts up a message in a box that tracks with the mouse. Its argument(s) are strings terminated by the argument (char *)0. Each string is placed on a separate line in the message box and is centered and displayed with the medium font. The message box replaces the mouse cursor and follows the mouse around the screen until a button is depressed.

If the first argument to *msgbox* is (char *)0, a message box with the message "No Memory" is displayed.

If there is no memory for displaying the given message, *msgbox* puts up a message box with the message "No Memory" and returns the value 0. Otherwise, it returns 1.

NAME

msgctl – message control operations

SYNOPSIS

#include <message.h>

int msgctl (msqid, cmd, buf)
long msqid;
int cmd;
struct msqid_ds *buf;

DESCRIPTION

Msgctl provides a variety of message control operations as specified by *cmd*. The following *cmds* are available:

IPC_STAT

Place the current value of each member of the data structure associated with *msqid* into the structure pointed to by *buf*.

IPC_SET

Set the value of the following members of the data structure associated with *msqid* to the corresponding value found in the structure pointed to by *buf:*

msg_qbytes
cid
state

The creator process id can be changed. This is done so that when this process is deleted, the queue will be deleted with it.

IPC_RMID

Remove the message queue identifier specified by *msqid* from the system and destroy the message queue and data structure associated with it. This can be executed by any process.

Msgctl will fail if one or more of the following are true:

Msqid is not a valid message queue identifier.

Cmd is not a valid command.

Cmd is equal to **IPC_SET** and an attempt is being made to increase the value of **msg_qbytes** over **MAX_QBYTES** (8192).

A side effect of *msgctl* is that it clears the MSG resource ready condition used by the *wait* and *own* resource functions.

Return Value

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned.

SEE ALSO

msgget(3L), msgop(3L), realtime(3R), resources(3R), structures(3R).

NAME

msgget – get message queue

SYNOPSIS

#include <message.h>

long msgget (key, msgflg)
long key;
int msgflg;

DESCRIPTION

Msgget returns the message queue identifier associated with *key*.

A message queue identifier and associated message queue and data structure are created for *key* if one of the following is true:

Key is equal to **IPC_PRIVATE**.

Key does not already have a message queue identifier associated with it, and (*msgflg* & **IPC_CREAT**) is "true".

Upon creation, the data structure associated with the new message queue identifier is initialized as follows:

Msg_qnum, **msg_curbytes**, **msg_list**, **msg_lspid**, **msg_lrpid**, **msg_stime**, and **msg_rtime** are set equal to 0.

Msg_ctime is set equal to the current time (*realtime*).

Msg_qbytes is set equal to **MAX_QBYTES**.

State is set to (**msgflg** & **NO_SAVE**).

Cid is set to the calling process's id (*P*).

Name is set to *key*.

If *state* is "true", the message queue will be deleted when the process *cid* is deleted.

Msgget will fail if one or more of the following are true:

A message queue identifier does not exist for *key* and (*msgflg* & **IPC_CREAT**) is "false".

A message queue identifier is to be created but there isn't enough memory.

A message queue identifier exists for *key* but ([*msgflg* & **IPC_CREAT**] && [*msgflg* & **IPC_EXCL**]) is "true".

Each message queue identifier that *msgget* returns is added to a list maintained for the application. This list defines which message queues constitute the MSG resource when using the *wait*(MSG) and *own*(MSG) functions.

Return Value

Upon successful completion, a non-negative long, namely a message queue identifier (pointer to msqid_ds structure), is returned. Otherwise, a value of -1 is returned.

SEE ALSO

msgctl(3L), msgop(3L), realtime(3R), resources(3R), structures(3R).

NAME

msgop – message operations

SYNOPSIS

#include <message.h>

int msgsnd (msqid, msgp, msgsz, msgflg)
long msqid;
struct msgbuf *msgp;
int msgsz, msgflg;

int msgrcv (msqid, msgp, msgsz, msgtyp, msgflg)
long msqid;
struct msgbuf [*]*msgp;
int msgsz;
long msgtyp;
int msgflg;

DESCRIPTION

Msgsnd is used to send a message to the queue associated with the message queue identifier specified by *msqid*. *Msgp* points to a structure containing the message. This structure is composed of the following members:

```
long    mtype;    /* message type */
char    mtext[];  /* message text */
```

Mtype is a positive integer that can be used by the receiving process for message selection (see *msgrcv* below). *Mtext* is any text of length *msgsz* bytes. *Msgsz* can range from 0 to what memory will allow.

Msgflg specifies the action to be taken if one or more of the following are true:

The number of bytes already on the queue is equal to **msg_qbytes**.

There is not enough memory to put the message on the queue.

These actions are as follows:

If (*msgflg* & **IPC_NOWAIT**) is "true", the message will not be sent and the calling process will return immediately.

If (*msgflg* & **IPC_NOWAIT**) is "false", the calling process will suspend execution until one of the following occurs:

The condition responsible for the suspension no longer exists, in which case the message is sent.

Msqid is removed from the system (see *msgctl*).

Msgsnd will fail and no message will be sent if one or more of the following are true:

Msqid is not a valid message queue identifier.

Mtype is less than 1.

The message cannot be sent for one of the reasons cited above and (*msgflg* & **IPC_NOWAIT**) is "true".

Msgsz is less than zero.

Upon successful completion, the following actions are taken with respect to the data structure associated with *msqid*.

Msg_qnum is incremented by 1.

Msg_lspid is set equal to the process id of the calling process.

Msg_stime is set equal to the current time.

An extra feature of *msgsnd* lets the user send messages without a copy being made. If (*msgflg* & **NO_COPY**) is "true", the message pointed to by *msgp* is put directly into the message queue. This can only be done if the *msgbuf* was allocated (created by a call to alloc). This is because the ownership of that memory must be changed. The sending process then no longer owns that memory and should not try to access it or free it.

Msgrcv reads a message from the queue associated with the message queue identifier specified by *msqid* and places it in the structure pointed to by *msgp*. This structure is composed of the following members:

```
long    mtype;     /* message type */
char    mtext[];   /* message text */
```

Mtype is the received message's type as specified by the sending process. *Mtext* is the text of the message. *Msgsz* specifies the size in bytes of *mtext*. The received message is truncated to *msgsz* bytes if it is larger than *msgsz* and (*msgflg* & **MSG_NOERROR**) is "true". The truncated part of the message is lost and no indication of the truncation is given to the calling process.

Msgtyp specifies the type of message requested as follows:

If *msgtyp* is equal to 0, the first message on the queue is received.

If *msgtyp* is greater than 0, the first message of type *msgtyp* is received.

If *msgtyp* is less than 0, the first message of the lowest type that is less than or equal to the absolute value of *msgtyp* is received.

Msgflg specifies the action to be taken if a message of the desired type is not on the queue. These are as follows:

If (*msgflg* & **IPC_NOWAIT**) is "true", the calling process will return

immediately with a return value of -1.

If (*msgflg* & **IPC_NOWAIT**) is "false", the calling process will suspend execution until one of the following occurs:

A message of the desired type is placed on the queue.

Msqid is removed from the system. When this occurs, a value of -1 is returned.

Msgrcv will fail and no message will be received if one or more of the following are true:

Msqid is not a valid message queue identifier.

Msgsz is less than 0.

Mtext is greater than *msgsz* and (*msgflg* & **MSG_NOERROR**) is "false".

The queue does not contain a message of the desired type and (*msgtyp* & **IPC_NOWAIT**) is "true".

Upon successful completion, the following actions are taken with respect to the data structure associated with *msqid*.

Msg_qnum is decremented by 1.

Msg_lrpid is set equal to the process id of the calling process.

Msg_rtime is set equal to the current time.

Again, the option of not copying the message is available. If (*msgflg* & **NO_COPY**) is "true", *msgp* is treated as a pointer to a pointer to a message. In other words, **msgp* is set to the address of the received message. The ownership of the message is then set to the receiving process. The receiving process can then read the message and free it.

A side effect of *msgrcv* is that it clears the MSG resource ready condition used by the *wait* and *own* resource functions.

Return Values

Upon successful completion, the return value is as follows:

Msgsnd returns a value of 0.

Msgrcv returns a value equal to the number of bytes actually placed into *mtext*.

Otherwise, a value of -1 is returned.

SEE ALSO

alloc(3R), msgctl(3L), msgget(3L), realtime(3R), resources(3R), structures(3R).

NAME

muldiv – calculate (a*b)/c accurately

SYNOPSIS

#include <dmd.h>
short muldiv (a, b, c)
int a, b, c;

DESCRIPTION

The *muldiv* function is a macro that returns the 16-bit result *(a*b)/c*. *(a*b)* is calculated to 32 bits to minimize the precision lost. The *muldiv* function is convenient for calculating transformations.

EXAMPLE

The following subroutine implements the transform(3R) function. It converts a point from window coordinates to screen coordinates:

```
#include <dmd.h>

Point
transform(p)
Point p;
{
        Point Do, Dc, ret;

        Do = Drect.origin;
        Dc = Drect.corner;

        ret.x = muldiv(p.x, Dc.x-Do.x, XMAX) + Do.x;
        ret.y = muldiv(p.y, Dc.y-Do.y, YMAX) + Do.y;

        return(ret);
}
```

The following subroutine does the opposite of the transform(3R) function. It converts a point from screen coordinates to window coordinates.

```
#include <dmd.h>

Point
untransform(p)
Point p;
{
        Point Do, Dc, ret;

        Do = Drect.origin;
        Dc = Drect.corner;

        ret.x = muldiv(p.x-Do.x, XMAX, Dc.x-Do.x);
        ret.y = muldiv(p.y-Do.y, YMAX, Dc.y-Do.y);
```

```
        return(ret);
    }
```

SEE ALSO

globals(3R), ptarith(3R), transform(3R/3L).

NAME

newrect – get swept or default rectangle

SYNOPSIS

#include <dmd.h>

Rectangle newrect (btn, rect)
int btn;
Rectangle rect;

DESCRIPTION

The *newrect* function prompts the user with a sweep cursor and an outline of the specified Rectangle *rect*, which then moves with the mouse. The function then busy waits for a Rectangle to be swept out with the specified *btn* button, or for the outline to be chosen by a click of *btn*. The function then returns the screen coordinates of the created Rectangle. If any button other than *btn* is depressed at any stage of the operation, *newrect* returns a null size Rectangle (i.e., the lower right corner is equal to the upper left origin). If *rect* is a null size Rectangle, no outline will be presented; only a sweep cursor will appear.

EXAMPLE

The following routine will permit the user to sweep out a Rectangle. As long as the Rectangle swept out does not have any points in common with *Drect*, the user will be prompted to sweep out another Rectangle. Once the swept out Rectangle has points within *Drect*, the Rectangle will be clipped to *Drect* and returned.

```
#include <dmd.h>

Rectangle
sweep_rect( )
{
        Rectangle r;

        do
                r = newrect(2, Drect);
        while (!rectclip(&r, Drect));
        return r;
}
```

SEE ALSO

box(3R), rectclip(3R).

DIAGNOSTICS

A mouse click is detected by the amount of mouse movement, not by the time between a button depression and release. The function *newrect* will see as a click any sweeping action whose result is a Rectangle less than 20 by 20 pixels.

NAME

norm, sqrtryz – return norm or coordinate of three-dimensional vector

SYNOPSIS

int norm (x, y, z)

int sqrtryz (r, y, z)

int r, x, y, z;

DESCRIPTION

The *norm* utility returns the norm of the vector (x, y, z). It is defined by the equation:

$$\sqrt{x^2+y^2+z^2}$$

The *sqrtryz* call returns the x-coordinate when passed the norm r and y and z coordinates. It is defined by the equation:

$$\sqrt{r^2-y^2-z^2}$$

NAME

peel – make process local and create new process

SYNOPSIS

#include <dmd.h>

Proc *peel (dstrect, flag)
Rectangle dstrect;
int flag;

DESCRIPTION

The *peel* function is an uncommon feature that allows a process to be made local and reshaped (or moved). The *peel* function then creates a new process that uses the previous process's window and host connection. The rectangle *dstrect* defines the new window (display) to be used by the peeled local process. The remaining window is associated with a new process that takes on the status of the connection of the previous process.

If argument *flag* is zero, the terminal's default process is started in the old window. If argument *flag* has a non-zero value, the process that is used to replace the peeled process is another invocation of the same process. The process must be cached to do this.

The peeled process will have its *P->state* variable updated. If the *dstrect* rectangle is the same size as the original window, both the MOVED and RESHAPED bits will be set; otherwise, just the RESHAPED bit is set. If the MOVED bit is set, the new window will have the contents of the old window copied into it. Otherwise, it is the responsibility of the peeled process to draw the new window's contents. The old window is always cleared.

The function will return -1 if it failed due to a lack of memory. It will return 0 if it failed because dstrect is smaller than 32x32. Otherwise, it will return a pointer to the newly created process running in the old window.

EXAMPLE

The following example shows how a process can peel itself and leave behind the default terminal process. The rectangle *dstrect* is specified through the mouse interface that allows the user to position or sweep out a rectangle using a default size as specified by the rectangle of the window or a new size using the mouse and button 2.

```
#include <dmd.h>
Proc *peel();
Rectangle newrect();

int
peelme()
{
    Proc *p;

    p = peel(newrect(2, display.rect), 0);
    if(p == (Proc *)-1)
        msgbox("no memory", (char *)0);
    else if(p == (Proc *)0)
```

```
        msgbox("swept window too small", (char *)0);
    return((long)p > 0);
}
```

SEE ALSO

cache(3L), local(3R), state(3R).

NAME

pfkey – get programmable function (PF) key strings

SYNOPSIS

int pfkey (keynum, str, maxlen)
int keynum, maxlen;
char *str;

DESCRIPTION

The *pfkey* function places up to (*maxlen*-1) characters from the given programmable function (PF) key *keynum* into the character array *str* which must be big enough to accept them. When *pfkey* encounters a null value or it has processed the (*maxlen*-1) th character, *pfkey* terminates the string with a NULL and returns the length of the string.

The value of *keynum* must be between 0x80 and 0x87, inclusive, which corresponds to PF key F1 to F8 on the keyboard. Each PF key can store up to 80 characters plus a NULL in the non-volatile BRAM space of the terminal.

EXAMPLE

The following code displays the value of programmable function key F1 on the window.

```
#include <dmd.h>

#define MAXPFKEY 80

showf1 () {
    char pfval[MAXPFKEY+1];

    pfkey (0x80, pfval, MAXPFKEY+1);
    jstring (pfval);
}
```

SEE ALSO

keyboard(3R).

NAME

point – draw a single pixel in a Bitmap

SYNOPSIS

#include <dmd.h>

void point (b, p, f)
Bitmap *b;
Point p;
Code f;

DESCRIPTION

The *point* function draws the pixel at Point *p* in the Bitmap **b* according to function Code *f*.

EXAMPLE

The following program determines the Point at the middle of the window and draws it. Hitting any key will exit the program.

```
#include <dmd.h>

Point middle();
Point add();
Point div();
Point sub();

main()
{
        Point centerdot;

        centerdot = middle();
        point (&display, centerdot, F_XOR);
        request(KBD);
        wait(KBD);
}

Point middle()
{
        Point center, offset;

        offset = div (sub (Drect.corner,
                Drect.origin), 2);
        center = add (Drect.origin, offset);

        return center;
}
```

SEE ALSO

jpoint(3R).

NAME

polygon: polyf, ptinpoly – polygon routines

SYNOPSIS

```
#include <dmd.h>

int polyf (bp, poly, np, t, f)

int ptinpoly (pt, poly, np)

Bitmap *bp;
Point poly[ ];
short np;
Texture16 *t;
Code f;
Point pt;
```

DESCRIPTION

The *polyf* function is used to fill a closed polygon defined by the *np* Points in the array of Points *poly*. The Points are absolute with respect to the Bitmap *bp*. The polygon is filled with the Texture16 *t* using Code *f*. The *polyf* call returns 0 if the polygon is filled and -1 if a memory allocation error occurred during processing, in which case the polygon is not filled.

The *ptinpoly* call determines whether the Point *pt* is contained in the polygon defined by *poly* and *np*. The *ptinpoly* function returns 1 if the Point is inside the polygon, 0 if the Point is not inside the polygon, and -1 if a memory allocation error occurs.

The polygon can consist of an arbitrary number of filled and unfilled regions. For example, a doughnut shape could be formed without filling the portion of the Bitmap corresponding to the hole of the doughnut. This permits a preservation of any background information previously placed in the Bitmap. Each region, filled or unfilled, is delimited by a Point whose x-coordinate has a value of *POLY_F* (defined in **dmd.h**). This Point in *poly* is ignored and merely serves as a flag indicating the start of a new region. There is always an assumed line connecting the first and last Point of each region.

EXAMPLE

The following code permits interactive drawing of a polygon with interior regions, fills it, and then uses *ptinpoly* to determine if the mouse is inside the polygon. Button 1 is used to draw the points, and button 2 sets a point to *POLY_F*. After the polygon fills, clicking button 1 with the mouse inside the polygon will ring the terminal bell. Clicking button 3 will exit the drawing and exit the program.

```
#include <dmd.h>

Point npoly[1000];

main()
{
        register int i, j;
        register char c;

        request (MOUSE|KBD);

        i = 0;
        for (;;)
        {
              while (i < 1000){
                    wait (MOUSE);
                    if( button1 () ){
                          npoly[i++] = mouse.xy;
                          point (&display, mouse.xy,
                              F_STORE);
                          sleep (10);
                          continue;
                    }
                    if( button2 () ){
                          npoly[i++].x = POLY_F;
                          continue;
                    }
                    if( button3 () ) break;
              }
              polyf (&display, npoly, i, &T_black,
                  F_STORE);

              wait(MOUSE);

              while( !button3() ){
                    wait (MOUSE);
                    if( button1() && ptinpoly
                        (mouse.xy, npoly, i) )
                          ringbell();

              }
              break;
        }
}
```

NAME

printf, fprintf, sprintf, lprintf, bprintf – print formatted output

SYNOPSIS

int Printf (format [, arg] ... **)**
void printf (format [, arg] ... **)**
char *format;

int Fprintf (stream, format [, arg] ... **)**
void fprintf (stream, format [, arg] ... **)**
char *format;
FILE *stream;

int Sprintf (s, format [, arg] ... **)**
void sprintf (s, format [, arg] ... **)**
char *s, format;

int Lprintf (format [, arg] ... **)**
void lprintf (format [, arg] ... **)**
char *format;

int Bprintf (format [, arg] ... **)**
void bprintf (format [, arg] ... **)**
char *format;

DESCRIPTION

Lprintf places output in the calling process's window. *Bprintf* places output at the bottom part of the 630's screen. *Printf* places output on the standard output stream *stdout*. *Fprintf* places output on the named output *stream*. *Sprintf* places "output," followed by the null character (**\0**), in consecutive bytes starting at **s*; it is the user's responsibility to ensure that enough storage is available. Each function returns the number of characters transmitted (not including the **\0** in the case of *Sprintf*), or a negative value if an output error was encountered. The *printf, lprintf, bprintf, fprintf,* and *sprintf* functions operate the same as those starting with a capital letter but support only a limited number of format options and no return value.

Each of these functions converts, formats, and prints its *arg*s under control of the *format*. The *format* is a character string that contains two types of objects: plain characters, which are simply copied to the output stream, and conversion specifications, each of which results in fetching of zero or more *arg*s. The results are undefined if there are insufficient *arg*s for the format. If the format is exhausted while *arg*s remain, the excess *arg*s are simply ignored.

Each conversion specification is introduced by the character %. After the %, the following appear in sequence:

Zero or more *flags*, which modify the meaning of the conversion specification.

An optional decimal digit string specifying a minimum *field width*. If the converted value has fewer characters than the field width, it will be padded on the left (or right, if the left-adjustment flag '–' (described below) has been given) to the field width. If the field width for an **s** conversion is preceded by a 0, the string is right

adjusted with zero-padding on the left.

A *precision* gives the minimum number of digits to appear for the **d**, **o**, **u**, **x**, or **X** conversions, the number of digits to appear after the decimal point for the **e** and **f** conversions, the maximum number of significant digits for the **g** conversion, or the maximum number of characters to be printed from a string in **s** conversion. The precision takes the form of a period (**.**) followed by a decimal digit string; a null digit string is treated as zero.

An optional **l** (ell) specifying that a following **d**, **o**, **u**, **x**, or **X** conversion character applies to a long integer *arg*. A **l** before any other conversion character is ignored.

A character that indicates the type of conversion to be applied.

A field width or precision may be indicated by an asterisk (*****) instead of a digit string. In this case, an integer *arg* supplies the field width or precision. The *arg* that is actually converted is not fetched until the conversion letter is seen, so the *arg*s specifying field width or precision must appear *before* the *arg* (if any) to be converted.

The flag characters and their meanings are:

- The result of the conversion will be left-justified within the field.

+ The result of a signed conversion will always begin with a sign (+ or -).

blank If the first character of a signed conversion is not a sign, a blank will be prefixed to the result. This implies that if the blank and + flags both appear, the blank flag will be ignored.

This flag specifies that the value is to be converted to an "alternate form." For **c**, **d**, **s**, and **u** conversions, the flag has no effect. For **o** conversion, it increases the precision to force the first digit of the result to be a zero. For **x** or **X** conversion, a non-zero result will have **0x** or **0X** prefixed to it. For **e**, **E**, **f**, **g**, and **G** conversions, the result will always contain a decimal point, even if no digits follow the point (normally, a decimal point appears in the result of these conversions only if a digit follows it). For **g** and **G** conversions, trailing zeroes will *not* be removed from the result (which they normally are).

The conversion characters and their meanings are:

d,o,u,x,X The integer *arg* is converted to signed decimal, unsigned octal, decimal, or hexadecimal notation (**x** and **X**), respectively; the letters **abcdef** are used for **x** conversion and the letters **ABCDEF** for **X** conversion. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeroes. (For compatibility with older versions, padding with leading zeroes may alternatively be specified by prepending a zero to the field width. This does not imply an octal value for the field width.) The default precision is 1. The result of converting a zero value with a precision of zero is a null string.

f The float or double *arg* is converted to decimal notation in the style "[-]ddd.ddd," where the number of digits after the decimal point is equal to the precision specification. If the precision is missing, six digits are output; if the precision is explicitly 0, no decimal point appears.

e,E The float or double *arg* is converted in the style "[-]d.ddd**e**±dd," where there is one digit before the decimal point and the number of digits after it is equal to the precision; when the precision is missing, six digits are produced; if the precision is zero, no decimal point appears. The **E** format code will produce a number with **E** instead of **e** introducing the exponent. The exponent always contains at least two digits.

g,G The float or double *arg* is printed in style **f** or **e** (or in style **E** in the case of a **G** format code), with the precision specifying the number of significant digits. The style used depends on the value converted: style **e** will be used only if the exponent resulting from the conversion is less than −4 or greater than the precision. Trailing zeroes are removed from the result; a decimal point appears only if it is followed by a digit.

c The character *arg* is printed.

s The *arg* is taken to be a string (character pointer) and characters from the string are printed until a null character (**0**) is encountered or the number of characters indicated by the precision specification is reached. If the precision is missing, it is taken to be infinite, so all characters up to the first null character are printed. A NULL value for *arg* will yield undefined results.

% Print a %; no argument is converted.

In no case does a non-existent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is simply expanded to contain the conversion result. Characters generated by *Printf*, *printf*, *Fprintf*, and *fprintf* are printed as if *putc*(3S) had been called. Characters generated by *Lprintf* and *lprintf* are printed as if *lputchar*(3L) had been called. Characters generated by *Bprintf* and *bprintf* are printed as if *bputchar*(3L) had been called.

The routines *printf*, *lprintf*, *bprintf*, *fprintf*, and *sprintf* support only the **d**, **o**, **u**, **x**, **s**, and **c** conversion characters and the optional **l**. They do not support floating point format, flags, field width, or precision options. This makes their code size small.

EXAMPLES

To print a date and time in the form "Sunday, July 3, 10:02," where *weekday* and *month* are pointers to null-terminated strings:

```
Lprintf("%s, %s %d, %d:%.2d", weekday, month, day, hour, min);
```

To print π to 5 decimal places:

```
Lprintf("pi = %.5f", 4 * atan(1.0));
```

SEE ALSO

jx(1), bputchar(3L), ecvt(3L), lputchar(3L).
putc(3S) in the *UNIX System V Programmer's Reference Manual.*

NAME

printq: printqempty, printqspace, printqclear – printer queue management

SYNOPSIS

int printqempty ( )

int printqspace ( )

void printqclear ( )

DESCRIPTION

Between terminal processes and the printer is a queue used for buffering characters being sent to the printer. The *printqempty* function returns 1 if this queue is empty. Otherwise, 0 is returned.

The *printqspace* function returns the number of characters that can be added to the queue before it fills. A full buffer affects the way the *psendchar* function operates.

The *printqclear* function removes all the characters currently in the queue. This prevents them from going to the printer. This routine clears the queue only if the printer is owned.

If the printer is not owned, *printqempty* returns 1 and *printqspace* returns 0.

SEE ALSO

psendchar(3R), resources(3R).

NAME

psendchar, psendnchars, xpsendchar, xpsendnchars – send character to printer port

SYNOPSIS

int psendchar (c)

int xpsendchar (c)

void psendnchars (n, s)

void xpsendnchars (n, s)

char c;
int n;
char *s;

DESCRIPTION

The *psendchar* function sends a single byte to the printer output queue. If the calling process does not own the printer, *psendchar* will send nothing to the printer and returns a 0. Otherwise, *psendchar* initiates printing and returns a 1.

The *xpsendchar* function is identical to *psendchar* except that it may expand tabs and filter escape sequences depending on the printer's setup values.

The *psendnchars* and *xpsendnchars* functions send the string *s* of length *n* to the printer. *Xpsendchars* handles expanding tabs and filtering escape sequences. These routines block until all the characters fit on the printer output queue.

If the printer output queue is full, the above functions will block the calling process until they are able to place the data into the printer queue.

SEE ALSO

resources(3R).

NAME

pt: Pt, Rpt, Rect – create a Point or Rectangle from arguments

SYNOPSIS

#include <dmd.h>

Point Pt (x, y)
int x, y;

Rectangle Rpt (p, q)
Point p, q;

Rectangle Rect (a, b, c, d)
int a, b, c, d;

DESCRIPTION

These functions are special macros that are to be used **only** in an argument list to a function. They are functionally equivalent to the ones in *fpt(3L)* but are faster for the above situation.

The *Pt* argument passes a coordinate pair as a Point to a function.

The *Rpt* argument passes two Points as a Rectangle to a function.

The *Rect* argument passes four coordinates (two coordinate pairs) as a Rectangle to a function.

EXAMPLE

The following subroutine draws two boxes in the upper left corner of the window.

```
#include <dmd.h>

drawboxes()
{
    Point add();
    Rectangle raddp();

    box(&display,
        Rpt(Drect.origin, add(Drect.origin, Pt(100,100))),
        F_STORE);

    box(&display,
        raddp(Rect(0,0,200,200), Drect.origin),
        F_STORE);
}
```

SEE ALSO

fpt(3L).

NAME

pt2win: point2window – find process table address of a window

SYNOPSIS

#include <dmd.h>
Proc *point2window (btn)
int btn;

DESCRIPTION

The *point2window* function implements the user interface for pointing to windows that are used by the Reshape, Move, Top, etc. functions of the 630 MTG main button 3 menu.

When *point2window* is called by an application program, the mouse cursor changes to a target icon. The function then busy waits for the user to move the mouse cursor into a window and press the button specified by *btn*. When this happens, the mouse cursor will change back to the previous mouse cursor, and the process table address of the window pointed to will be returned.

The function will return (Proc *)0 if the user either clicks button *btn* while the target is over background texture or clicks a mouse button other than button *btn*.

EXAMPLE

The follow subroutine will flash a window pointed to with button 3. Note that in the following code, *p->layer* is equivalent to *&display* for process p and that *p->rect* is equivalent to *Drect* for process p.

```
#include <dmd.h>

flash()
{
        Proc *p;
        Proc *point2window();

        p = point2window(3);
        if(p) {
                rectf(p->layer, p->rect, F_XOR);
                nap(20);
                rectf(p->layer, p->rect, F_XOR);
        }
}
```

SEE ALSO

globals(3R), rectf(3R), sleep(3R).

NAME

ptarith: add, sub, mul, div – arithmetic on Points

SYNOPSIS

#include <dmd.h>

Point add (p, q)

Point sub (p, q)

Point mul (p, a)

Point div (p, a)

Point p, q;
int a;

DESCRIPTION

The *add* function returns the Point sum of its arguments:

```
{ p.x+q.x, p.y+q.y }
```

The *sub* function returns the Point difference of its arguments:

```
{ p.x-q.x, p.y-q.y }
```

The *mul* function returns the Point:

```
{ p.x*a, p.y*a }
```

The *div* function returns the Point:

```
{ p.x/a, p.y/a }.
```

EXAMPLE

The following routine returns the center Point of a *window*.

```
#include <dmd.h>
Point add();
Point sub();
Point div();
Point
getcenter()
{
    Point offset;
    offset = div (sub
        (Drect.corner,Drect.origin), 2);
    return add (Drect.origin, offset);
}
```

SEE ALSO
globals(3R), rectarith(3R), transform(3R/3L).

NAME

ptinrect – check for Point inclusion in a Rectangle

SYNOPSIS

 # include <dmd.h>

int ptinrect (p, r)
Point p;
Rectangle r;

DESCRIPTION

The *ptinrect* function returns 1 if *p* is a Point within Rectangle *r* ; and 0, otherwise.

EXAMPLE

The following routine will draw a box in the middle of a window. Then when button 1 is depressed within the box, the bell will ring. The routine returns when a key is typed.

```
#include <dmd.h>
Point add();
Point sub();
Point div();

ringbox()
{
   Point center, offset;
   Rectangle midbox;

   offset = div (sub (Drect.corner,
                Drect.origin), 2);
   center = add (Drect.origin, offset);
   midbox.origin = sub (center, Pt (16, 16));
   midbox.corner = add (center, Pt (16, 16));
   box (&display, midbox, F_XOR);
   request (MOUSE|KBD);
   while (kbdchar () == -1){
      wait (MOUSE);
      if (ptinrect (mouse.xy, midbox) &&
            button1()) ringbell();
   }
}
```

NAME

qsort – quicker sort

SYNOPSIS

void qsort ((char *) base, nel, sizeof (*base), compar)
unsigned nel;
int (*compar)( );

DESCRIPTION

qsort is an implementation of the quicker-sort algorithm. It sorts a table of data in place.

The *base* points to the element at the base of the table. The pointer to the base of the table should be of type pointer-to-element, and cast to type pointer-to-character.

The *nel* argument is the number of elements in the table.

The *compar* element is the name of the comparison function which is called with two arguments that point to the elements being compared. It need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared. The function must return an integer less than, equal to, or greater than zero based on whether the first argument is to be considered less than, equal to, or greater than the second.

EXAMPLE

The following routine will accept Points designated by clicking button 1 and then sort the array based on *x* and *y* coordinate values.

```
#include <dmd.h>

Point p[100];

ptcompar(p, q)
Point *p, *q;
{
      if ( p->y < q->y ) return -1;
      if ( p->y == q->y ){
            if ( p->x == q->x ) return 0;
            if ( p->x < q->x ) return -1;
            return 1;
      }
      return 1;
}

Point *
sortpoints()
{
      int i;

      request (MOUSE|KBD);
      for( i = 0; i < 100;){
            wait (MOUSE);
            if( button1 () ){
```

```
                p[i++] = mouse.xy;
                sleep (10);
                continue;
            }
            if( button3 () ) break;
        }
        qsort ((char *)p, i, sizeof(Point),
                ptcompar);
        return p;
    }
```

NOTES

The order in the output of two items which compare as equal is unpredictable.

SEE ALSO

bsearch(3L), lsearch(3L), str(3L).

NAME

rand, srand – simple random-number generator

SYNOPSIS

int rand ( )

void srand (seed)
unsigned seed;

DESCRIPTION

Rand uses a multiplicative, congruential, random-number generator with period 2^{32} that returns successive pseudo-random numbers in the range from 0 to $2^{15}-1$.

Srand can be called at any time to reset the random-number generator to a random starting point. The generator is initially seeded with a value of 1.

NOTES

The spectral properties of *rand* are limited. *Drand48*(3L) provides a much better, though more elaborate, random-number generator.

SEE ALSO

drand48(3L).

NAME

rcvchar – receive character from host

SYNOPSIS

int rcvchar ( )

DESCRIPTION

The *rcvchar* function returns the next character received from the host. If there are no characters available, *rcvchar* returns -1.

Since local programs have no host connection, a call to *rcvchar* by a local program will always return -1.

EXAMPLE

The following program is a very simple terminal emulator.

```
#include <dmd.h>

main()
{
        int c;

        request(RCV);
        for(;;) {
                wait(RCV);
                while( (c=rcvchar()) != -1 )
                        lputchar(c);
        }
}
```

SEE ALSO

local(3R), resources(3R), sendchar(3R).

NAME

realtime – terminal clock

SYNOPSIS

unsigned long realtime ( )

DESCRIPTION

The *realtime* function returns the number of 60Hz clock ticks since the terminal was booted.

EXAMPLE

The following code could be used to roughly measure the performance of *bitblt*.

```
#include <dmd.h>
unsigned long realtime();

unsigned long
perf_bitblt( sb, r, db, p, f)
Bitmap *sb, *db;
Rectangle r;
Point p;
Code f;
{
        unsigned long begin;

        begin = realtime ();
        bitblt (sb, r, db, p, f);
        return (realtime () - begin);
}
```

NAME

rectarith: raddp, rsubp – arithmetic on Rectangles

SYNOPSIS

#include <dmd.h>
Rectangle raddp (r, p)

Rectangle rsubp (r, p)

Rectangle r;
Point p;

DESCRIPTION

The *raddp* function returns the Rectangle defined by (**add(r.origin, p)**, **add(r.corner, p)**).

The *rsubp* function returns the Rectangle defined by (**sub(r.origin, p)**, **sub(r.corner, p)**).

EXAMPLE

The following code will return a rectangle with the mouse coordinates at the center of the box.

```
#include <dmd.h>

Rectangle
makebox()
{
return (raddp (Rect(-15, -15, 15, 15),
          mouse.xy));

}
```

SEE ALSO

ptarith(3R), structures(3R).

NAME

rectclip – clip a Rectangle to another Rectangle

SYNOPSIS

#include <dmd.h>

int rectclip (rp, s)
Rectangle *rp, s;

DESCRIPTION

The *rectclip* function clips the Rectangle pointed to by *rp* so that it is completely contained within the Rectangle *s*. The return value is 1 if any part of **rp* is within *s*. Otherwise, the return value is 0 and **rp* is unchanged.

EXAMPLE

See the example for *newrect*(3R).

SEE ALSO

newrect(3R).

NAME

rectf – perform function on Rectangle in Bitmap

SYNOPSIS

```
#include <dmd.h>

void rectf (b, r, f)
Bitmap *b;
Rectangle r;
Code f;
```

DESCRIPTION

The *rectf* function performs the action specified by the function Code *f* on the Rectangle *r* within the Bitmap **b*.

EXAMPLE

The following routine will "doodle" on the screen using a Rectangle.

```
#include <dmd.h>

int request();
int wait();
Point add();
void rectf();

main()
{
    Rectangle r;
    Point s;

    s.x = 16;
    s.y = 16;
    request (MOUSE);
    for (wait(MOUSE); !button3();
      wait(MOUSE)){
        r.origin = mouse.jxy;
        r.corner = add (r.origin, s);
        if( button1() )
            rectf (&display, r, F_STORE);
        if( button2() )
            rectf (&display, r, F_CLR);
    }
}
```

SEE ALSO

jrectf(3R), structures(3R).

NAME

rectxrect: rectXrect – check for overlapping Rectangles

SYNOPSIS

#include <dmd.h>

int rectXrect (r, s)
Rectangle r, s;

DESCRIPTION

The *rectXrect* function returns 1 if *r* and *s* share any point; 0, otherwise.

EXAMPLE

The following routine will prompt the user to sweep out two Rectangles with button 1. If they intersect, the bell will ring. Typing any key will exit the program.

```
#include <dmd.h>

int request();
int wait();
void box();
void ringbell();
Rectangle newrect();

main()
{
  Rectangle r, s;

  request (MOUSE|KBD);
  while (kbdchar () == -1) {
      wait (MOUSE);

      r = newrect (1, Rect(0,0,0,0));
      box (&display, r, F_XOR);
      s = newrect (1, Rect(0,0,0,0));
      box (&display, s, F_XOR);
      if (rectXrect (r, s))
         ringbell ();
      box (&display, r, F_XOR);
      box (&display, s, F_XOR);

      wait (MOUSE | KBD);
  }
}
```

SEE ALSO

structures(3R).

NAME

resources: request, own, wait, alarm – routines dealing with resources

SYNOPSIS

```
#include <dmd.h>

int request (r)

int own ( )

int wait (r)

void alarm (t)

int r;
unsigned t;
```

DESCRIPTION

The above routines deal with the following 630 MTG resources:

KBD	characters received from the 630 MTG keyboard
SEND	send characters from the 630 MTG to the host
MOUSE	mouse buttons and cursor position
RCV	characters received by 630 MTG from host process
PSEND	send characters to the printer
CPU	630 MTG cpu
ALARM	alarm has "fired"
RESHAPED	window has been reshaped or moved
DELETE	application is being deleted
MSG	state of message queues has changed

Request announces a program's intent to use one or more resources and is usually called once early in the program. The **r** argument is a bit vector indicating which resources are being requested. Compose **r** by using the bit-wise inclusive OR operator of the above resources (MOUSE¦KBD means you are referring to the mouse and keyboard resources). *Request* returns a bit vector that indicates those resources for which the request succeeded.

Note that if a program calls *request* several times, each *request* overrides all previously *request*ed resources. This means that a resource previously *request*ed, but not specified in the latest call to *request* , will no longer be available to the application program.

If the keyboard is not requested, characters typed will be sent to the host. If the mouse is not requested, mouse events in the application's window will be interpreted by the terminal's control process (the terminal's control process is the process that displays the main button 3 menu and makes windows top and current when pointed to by button 1). SEND and CPU are always implicitly *request*ed.

A request of PSEND will fail if another application program has already requested it. Currently, PSEND is the only resource that may fail a request. PSEND must be requested and owned before sending characters to the printer.

Requesting the DELETE resource tells the terminal's control process to not allow a user to delete the applications window from the main button 3 menu. Rather, the control process sets a flag which causes *own()&DELETE* to be true. This is intended for use by applications which need to perform some type of cleanup before being deleted. When *own()&DELETE* becomes true, it is the responsibility of the application to perform its cleanup and delete itself (see example below).

The *own* function returns a bit vector indicating which resources are ready to be serviced.

The *wait* function suspends the application, enabling others to run, until at least one of the resources in the bit vector *r* is ready for service. The return value is a bit vector indicating which resources are ready for service. Applications wishing to give up the processor to enable other applications to run may call *wait*(**CPU**). In this case, *wait* will always return as soon as all other applications have had a chance to run.

The *alarm* function starts a timer which will "fire" **t** ticks (60ths of a second) in the future. Calling *alarm* implicitly *request*s the **ALARM** . The resource **ALARM** can be used to check the status of the timer. The *own*( ) **&ALARM** call will indicate whether or not the alarm timer has fired. A *wait*(ALARM) call allows the application to give up the CPU until the specified number of ticks has elapsed. An *alarm*(0) call cancels a previous call to *alarm*.

An *alarm* call does not interfere with *sleep* and vice versa.

The call *wait* (RESHAPED) will suspend the application until the application's window has been either moved or reshaped. The MOVED and RESHAPED bits of the application's state variable (*P->state*) can be read to determine which of the two has occurred. If the window was moved, both the MOVED and RESHAPED bits are set; only the RESHAPED bit is set if the window was reshaped. The application is responsible for clearing the state variable MOVED and RESHAPED bits regardless of whether they are read or not. Subsequent *wait* (RESHAPED) calls will return immediately if the application has not cleared the MOVED and RESHAPED bits. RESHAPED is always implicitly requested.

The call *wait* (MSG) will suspend the application until the state of any of the message queues associated with the application changes. A list of message queue identifiers is maintained for each application. A message queue identifier is entered into this list during a call by the application to the function msgget when either a message queue is created or an identifier for an existing message queue is retrieved. Once added to this list, a message queue identifier is removed from the list only when the message queue is deleted. The message queue can be deleted by any running application with a call to the function msgctl or is deleted automatically if it was created with the NO_SAVE option and the creating application is deleted or exits.

The function calls *own*(MSG) and *wait*(MSG) will return the MSG bit true in the returned bit vector after any of three state changes occur to any of the message queues associated with the application: 1) a message queue when added to the application's list already has one or more messages on it, 2) a message is received at a message queue, or 3) a message queue on the

list is deleted. This condition remains true until cleared by a call to the function msgctl to examine a message queue or by a call to the function msgrcv to receive a message from a queue. Note that several applications may simultaneously be waiting for the same message queue. If a message arrives at the queue, all of the waiting applications will be restarted. If the first application that is restarted removes the arrived message from the queue and does not replace it back on the queue with a call to the function msgsnd, the other applications that were waiting will find no message when they are restarted. Similarly, an application may find no message queue when it is restarted if the queue was deleted by another application. If an application sends a message to a message queue that is on its own message queue list, the wait condition becomes true for it also.

EXAMPLE

The following program fragment shows how an application can give up button 3 processing to the terminal's control process. This is how applications who request the mouse but do not use button 3 can let the main button 3 menu be displayed.

Note that the **sleep**(2) below is necessary because the terminal's control process runs only once every tick (60th of a second) of the realtime clock.

```
#include <dmd.h>

main()
{
        int r;

        r = request (MOUSE);
        if (button3()){
           request (r & ~MOUSE);  /* release the mouse */
           sleep (2);/* sleep(2) since control process */
             /* only runs once every clock tick */
           request (r);
        }
}
```

The following program shows how to use the DELETE resource. This program will ring the terminals bell before it is deleted. Typically, rather than ringing the bell, some application specific cleanup would be performed.

```
#include <dmd.h>

main()
{
        request(DELETE|MOUSE);

        for(;;) {
              wait(DELETE|MOUSE);
              if(own()&DELETE) {
                    ringbell(); /* or cleanup */
                    delete();  /* delete me */
```

```
            }
        }
    }
```

The following code fragment shows how to use the RESHAPED resource. Upon return from *wait* it tests the state variable to determine if the window was moved or reshaped. Depending on which occurred, an appropriate flag is set and the state variable is cleared.

```
#include <dmd.h>
    .
    .
    int moved=0;
    int reshaped=0;
    .
    .
    wait(RESHAPED);
    if (P->state&MOVED) {
       moved++;
       P->state &= ~(MOVED|RESHAPED);
    }
    else if ((P->state&RESHAPED) && !(P->state&MOVED))
    { reshaped++;
       P->state &= ~RESHAPED;
    }
    .
    .
```

SEE ALSO

msgctl(3L), msgget(3L), msgop(3L), sleep(3R), state(3R).

NAME

ringbell, click – ring, click the 630 MTG

SYNOPSIS

void ringbell ( )

void click ( )

DESCRIPTION

The *ringbell* function rings the bell on the 630 MTG.

The *click* function makes the 630 MTG click as if a key has been depressed.

EXAMPLE

See the example in *polygon*(3L).

NAME

rol, ror – rotate bits

SYNOPSIS

int rol (x, n)

int ror (x, n)

int x, n;

DESCRIPTION

The *rol* function returns *x* logically bit-rotated left by *n*.

The *ror* function returns *x* logically bit-rotated right by *n*.

EXAMPLE

The following subroutine can be used to determine whether a Point *p* in a Bitmap *b* is on or off (returning 1 or 0, respectively).

```
#include <dmd.h>

pixel (b, p)
Bitmap *b;
Point p;
{
        Word *w;

        w = addr (b, p);
        return ( (rol (*w, p.x%WORDSIZE)
                & FIRSTBIT)==FIRSTBIT);
}
```

This routine is implemented differently in *addr*(3R).

SEE ALSO

addr(3R).

NAME

screenswap – swap screen Rectangle and Bitmap

SYNOPSIS

#include <dmd.h>

void screenswap (b, r, s)
Bitmap *b;
Rectangle r, s;

DESCRIPTION

The *screenswap* function does an in-place exchange of the Rectangle *r* within the Bitmap *b* and screen rectangle *s*. This exchange is done by *bitblt*'ing the bitmaps back and forth three times in XOR mode. This technique allows bitmaps to be exchanged without need for intermediate storage.

The action of screenswap is undefined, if *r* and *s* are not the same size.

Screenswap writes to the *physical* bitmap, so the *s* argument is clipped to the screen, not to the window's rectangle (display.rect).

EXAMPLE

The following program floats a picture of a sailboat across the screen.

The 630 MTG mouse cursor is painted in XOR mode and therefore changes to inverse video when it moves over highlighted areas. This sailboat, however, does not inverse video when it moves over parts of the screen that are highlighted. This is accomplished by saving whatever is on the screen in the spot where the sailboat is currently painted and restoring the screen when the sailboat moves. The unique aspect of this is that the same physical memory is used to alternately store the picture of the sailboat and the saved screen rectangle, and these two bitmaps are swapped without use of intermediate storage.

```
#include <dmd.h>

unsigned short sailicon[] = {
        0xFDFF, 0xF9FF, 0xF1FF, 0xE0FF,
        0xFD7F, 0xF9BF, 0xF5DF, 0xEDEF,
        0xDDF7, 0xDDF7, 0xBDFB, 0xB8FB,
        0x0000, 0x8003, 0xE007, 0xFFFF,
};

Bitmap sailmap = {
        (Word *)sailicon,
        1,
        (short)0, (short)0, (short)16, (short)16,
        (char *)0
};

extern Rectangle fRpt();
extern Rectangle raddp();

main()
```

```
{
        Rectangle r;

        r = fRpt(0, YMAX/2-8, 16, YMAX/2+8);

        /* put the sailboat onto the screen */
        screenswap(&sailmap, sailmap.rect, r);

        /* move the sailboat across the screen */
        while(r.corner.x <= YMAX) {
                sleep(3);
                screenswap(&sailmap, sailmap.rect, r);
                r = raddp(r, Pt(1,0));
                screenswap(&sailmap, sailmap.rect, r);
        }

        /* remove the sailboat for the last time */
        screenswap(&sailmap, sailmap.rect, r);
}
```

This example is very similar to how message boxes are implemented. Message boxes are rectangles containing messages which float around the screen when the mouse moves.

SEE ALSO

bitblt(3R), msgbox(3R).

NAME

segment – draw a line segment in a Bitmap

SYNOPSIS

#include <dmd.h>

void segment (b, p, q, f)
Bitmap *b;
Point p, q;
Code f;

DESCRIPTION

The *segment* utility draws a line segment in Bitmap *b* from Point *p* to Point *q* with function Code *f*.

Like all the other graphics operations, *segment* clips the line so that only the portion of the line intersecting the Bitmap is displayed.

EXAMPLE

The following call simply draws a line connecting a window's origin Point to its corner Point (a diagonal line from the upper left corner to the lower right corner of the window).

```
#include <dmd.h>

connectcorners()
{

        segment (&display, Drect.origin,
                Drect.corner, F_XOR);

}
```

SEE ALSO

jsegment(3R).

NAME

sendchar, sendnchars – send character(s) to host

SYNOPSIS

int sendchar (c)
char c;

void sendnchars (n, p)
int n;
char *p;

DESCRIPTION

The *sendchar* function sends a single byte to the host which will normally be read on the standard input of the host process. The bytes sent will be processed as though they were typed on the keyboard by the user.

Since local programs have no host connection, a call to *sendchar* by a local program will always return -1. Sendchar will always return 1 for non-local programs.

The *sendnchars* function is similar to *sendchar* except *n* characters pointed to by *p* are sent to the host.

A call to *sendchar* or *sendnchars* by an application program causes the bytes to be sent to the host computer to be placed into a buffer in the terminal. The terminal will then send the bytes in the buffer to the host as fast as the communication line to the host is able to transmit. An application program is generally able to queue requests to send bytes to the host much faster than the terminal is able to actually send the bytes over the communication line to the host. So, if an application rapidly sends many bytes to the host, internal terminal buffers eventually fill up. When this happens, the *sendchar* and *sendnchars* routines will call *wait* to block the calling process until internal buffers are no longer full. This whole process is transparent to applications, but application program writers may want to be aware that a call to *sendchar* or *sendnchars* may not return immediately.

EXAMPLE

The following program will send a *ls* command to the host and display its output. Typing a 'q' on the keyboard will cause the program to exit.

```
#include <dmd.h>

main()
{
        int c;

        request(SEND|RCV|KBD);
        sendnchars(3, "ls\r");
        do {
                wait(RCV|KBD);
                while( (c=rcvchar()) != -1 )
                        lputchar(c);
        } while(kbdchar() != 'q');
}
```

SEE ALSO
local(3R), rcvchar(3R), resources(3R).

NAME

setled: setLEDcap, setLEDscr – set the caps lock and scroll lock LEDs

SYNOPSIS

void setLEDcap (n)

void setLEDscr (n)

int n;

DESCRIPTION

The *setLEDcap* and *setLEDscr* functions turn on and off the caps lock and scroll lock LEDs, respectively. If *n* is 1, the LED is turned on. If *n* is 0, the LED is turned off.

If the window is not current when the function is called, the state of the LED is remembered and automatically changed when the window is made current.

These functions also update the label area to reflect the status of the LEDs if the label is being used. The *setLEDscr* function will update the SCR_LOCK bit of P->state.

These functions should not be used unless the program is using the keyboard in NOTRANSLATE mode. Otherwise, the LEDs may not match the actual state of the caps lock and scroll lock keys.

SEE ALSO

keyboard(3R).

NAME

setupval – return a setup option

SYNOPSIS

#include <setup.h>

int setupval (obj, opt)
int obj, opt;

DESCRIPTION

The *setupval* function returns the value of a setup option. The first argument, *obj*, is an object that has setup values. Possible objects are the terminal (*S_PREF*), the printer (*S_PRINT*), and the host (*whathost()*). The second argument, *opt*, is a particular option for that object. The return value is an integer which describes that setting for that option. This is summarized in the table below.

Object	Option	Returned	Meaning
S_PREF (user's preferences)	S_PREF_CTRL (control character display)	S_PREF_CTRL_VIS	visible
		S_PREF_CTRL_INVIS	invisible
		S_PREF_CTRL_SPACE	a space
	S_PREF_KCLK (keyboard click)	S_PREF_KCLK_OFF	off
		S_PREF_KCLK_ON	on
	S_PREF_KVOL (keyboard volume)	S_PREF_KVOL_0	off
		S_PREF_KVOL_1	1
		S_PREF_KVOL_2	2
		S_PREF_KVOL_3	3
		S_PREF_KVOL_4	4
		S_PREF_KVOL_5	5
		S_PREF_KVOL_6	6
		S_PREF_KVOL_7	full
	S_PREF_CURS (cursor mode)	S_PREF_CURS_NOBLK	no blinking
		S_PREF_CURS_BLK	blinking
	S_PREF_KRPT (keyboard repeat rate)	S_PREF_KRPT_15	15 per second
		S_PREF_KRPT_20	20 per second
		S_PREF_KRPT_30	30 per second
		S_PREF_KRPT_60	60 per second
	S_PREF_WBUF (windowproc buffer)	S_PREF_WBUF_OFF	off
		S_PREF_WBUF_ON	on
	S_PREF_WTYP (windowproc type)	S_PREF_WTYP_BASIC	basic
		S_PREF_WTYP_EHN	enhanced

S_PRINT (printer)	S_PRINT_TAB (tab expansion)	S_PRINT_TAB_NO	no
		S_PRINT_TAB_YES	yes
	S_PRINT_ESC (filter escapes)	S_PRINT_ESC_NO	no
		S_PRINT_ESC_YES	yes
whathost() (process's host)	S_HOST_ENC (encoding)	S_HOST_ENC_OFF	off
		S_HOST_ENC_ON	on
	S_HOST_RTN (sent return key definition)	S_HOST_RTN_CR	carriage return
		S_HOST_RTN_LF	line feed
		S_HOST_RTN_CRLF	carriage return and line feed
	S_HOST_NL (newline definition)	S_HOST_NL_LF	line feed
		S_HOST_NL_CRLF	carriage return and line feed
	S_HOST_FONT (font)	S_HOST_FONT_SMALL	smallfont
		S_HOST_FONT_MEDIUM	mediumfont
		S_HOST_FONT_LARGE	largefont
	S_HOST_COL	multiplexed columns	number
	S_HOST_ROW	multiplexed rows	number
	S_HOST_NCOL	nonmultiplexed columns	number
	S_HOST_NROW	nonmultiplexed rows	number
	S_HOST_FIXED (fixed size window)	S_HOST_FIXED_NO	no
		S_HOST_FIXED_YES	yes

If the object is invalid, *setupval* returns -1. If the option for a given object is invalid, the return value is undefined.

EXAMPLE

The following example prints out the setting of some options.

```
#include <setup.h>

printval()
{
    int i;

    lprintf("key click is ");
    switch(setupval(S_PREF, S_PREF_KCLK))
    {
        case S_PREF_KCLK_OFF: lprintf("off\n"); break;
        case S_PREF_KCLK_ON:  lprintf("on\n"); break;
    }
    i = setupval(whathost(),
```

```
                        ismpx()? S_HOST_ROW: S_HOST_NROW);
        if(i == -1)
            lprintf("I am local\n");
        else
            lprintf("default rows = %d\n", i);
    }
```

SEE ALSO

ismpx(3R), whathost(3R).

NAME

sleep, nap – suspend program execution

SYNOPSIS

void sleep (nticks)

void nap (nticks)

unsigned int nticks;

DESCRIPTION

The *nap* function busy loops for *nticks* ticks of the 60 Hz internal clock. To avoid interfering with screen refresh, programs drawing rapidly changing scenes should *nap* for a couple ticks between updates to synchronize the display and memory.

The *sleep* function is identical to *nap* except that it gives up the processor for the interval. *Sleep* should be used in preference to *nap* unless there is some reason why other applications should not be allowed to run. A process that never calls *wait* or *sleep* can lock out all other 630 MTG processes.

SEE ALSO

resources(3R).

NAME

ssignal, gsignal – software signals

SYNOPSIS

#include <ccs/signal.h>

int (*ssignal (sig, action))( )
int sig, (*action)( );

int gsignal (sig)
int sig;

DESCRIPTION

ssignal and *gsignal* implement a software facility similar to *signal*(2). This facility is used by the Standard C Library to enable users to indicate the disposition of error conditions, and it is also made available to users for their own purposes.

Software signals made available to users are associated with integers in the inclusive range 1 through 16. A call to *ssignal* associates a procedure, *action*, with the software signal *sig*; the software signal, *sig*, is raised by a call to *gsignal*. Raising a software signal causes the action established for that signal to be *taken*.

The first argument to *ssignal* is a number identifying the type of signal for which an action is to be established. The second argument defines the action; it is either the name of a (user-defined) *action function* or one of the manifest constants **SIG_DFL** (default) or **SIG_IGN** (ignore). *ssignal* returns the action previously established for that signal type; if no action has been established or the signal number is illegal, *ssignal* returns **SIG_DFL**.

Gsignal raises the signal identified by its argument, *sig*:

If an action function has been established for *sig*, then that action is reset to **SIG_DFL** and the action function is entered with argument *sig*. *Gsignal* returns the value returned to it by the action function.

If the action for *sig* is **SIG_IGN**, *gsignal* returns the value 1 and takes no other action.

If the action for *sig* is **SIG_DFL**, *gsignal* returns the value 0 and takes no other action.

If *sig* has an illegal value or no action was ever specified for *sig*, *gsignal* returns the value 0 and takes no other action.

Notes

There are some additional signals with numbers outside the range 1 through 16 which are used by the Standard C Library to indicate error conditions. Thus, some signal numbers outside the range 1 through 16 are legal, although their use may interfere with the operation of the Standard C Library.

SEE ALSO

signal(2) in the *UNIX System V Programmer's Reference Manual.*

NAME

state: P->state, MOVED, RESHAPED, NO_RESHAPE - per process windowing states

SYNOPSIS

#include <dmd.h>

long P->state;

DESCRIPTION

P->state is the state variable for an application running in the 630 MTG.

The fields in the state variable relevant to windowing operations are:

MOVED

RESHAPED

NO_RESHAPE

These bits are set or checked by the terminal during a windowing operation.

If an application program wants to do some special processing after being moved or reshaped, it must periodically check the proper *state* bits. When the pertinent event occurs, the application program should perform its special processing and then reset the proper bits in the state variable as follows:

For a process to test whether its window has been moved , it must check:

(P->state&MOVED)

To reset the MOVED condition it is necessary to execute:

P->state &= ~(MOVED|RESHAPED);

For a process to test whether its window has been reshaped, it must check:

(P->state&RESHAPED && !(P->state&MOVED))

To reset the RESHAPED condition it is necessary to execute:

P->state &= ~RESHAPED;

The reason for the interaction of the MOVED and RESHAPED bits is purely historical.

The NO_RESHAPE bit should be set if the application program does not want to be reshaped by the user through the global mouse operation. A message box with the message "Fixed size window: Cannot be reshaped" will be displayed if the user attempts to reshape the window. Clearing the bit makes the window reshapable again.

SEE ALSO

keyboard(3r).

NAME

str: strcat, strncat, strcmp, strncmp, strcpy, strncpy, strlen, strchr, strrchr, strpbrk, strspn, strcspn, strtok – string operations

SYNOPSIS

```
#include <ccs/string.h>

char *strcat (s1, s2)
char *s1, *s2;

char *strncat (s1, s2, n)
char *s1, *s2;
int n;

int strcmp (s1, s2)
char *s1, *s2;

int strncmp (s1, s2, n)
char *s1, *s2;
int n;

char *strcpy (s1, s2)
char *s1, *s2;

char *strncpy (s1, s2, n)
char *s1, *s2;
int n;

int strlen (s)
char *s;

char *strchr (s, c)
char *s;
int c;

char *strrchr (s, c)
char *s;
int c;

char *strpbrk (s1, s2)
char *s1, *s2;

int strspn (s1, s2)
char *s1, *s2;

int strcspn (s1, s2)
char *s1, *s2;

char *strtok (s1, s2)
char *s1, *s2;
```

DESCRIPTION

The arguments *s1, s2* and *s* point to strings (arrays of characters terminated by a null character). The functions *strcat, strncat, strcpy,* and *strncpy* all alter *s1*. These functions do not check for overflow of the array pointed to by *s1*.

Strcat appends a copy of string *s2* to the end of string *s1*. *Strncat* appends at most *n* characters. Each returns a pointer to the null-terminated result.

Strcmp compares its arguments and returns an integer less than, equal to, or greater than 0, if *s1* is lexicographically less than, equal to, or greater than *s2*. *Strncmp* makes the same comparison but looks at most *n* characters.

Strcpy copies string *s2* to *s1*, stopping after the null character has been copied. *Strncpy* copies exactly *n* characters, truncating *s2* or adding null characters to *s1* if necessary. The result will not be null-terminated if the length of *s2* is *n* or more. Each function returns *s1*.

Strlen returns the number of characters in *s*, not including the terminating null character.

Strchr (*strrchr*) returns a pointer to the first (last) occurrence of character *c* in string *s*, or a NULL pointer if *c* does not occur in the string. The null character terminating a string is considered to be part of the string.

Strpbrk returns a pointer to the first occurrence in string *s1* of any character from string *s2*, or a NULL pointer if no character from *s2* exists in *s1*.

Strspn (*strcspn*) returns the length of the initial segment of string *s1* which consists entirely of characters from (not from) string *s2*.

Strtok considers the string *s1* to consist of a sequence of zero or more text tokens separated by spans of one or more characters from the separator string *s2*. The first call (with pointer *s1* specified) returns a pointer to the first character of the first token, and will have written a null character into *s1* immediately following the returned token. The function keeps track of its position in the string between separate calls, so that subsequent calls (which must be made with the first argument a NULL pointer) will work through the string *s1* immediately following that token. In this way, subsequent calls will work through the string *s1* until no tokens remain. The separator string *s2* may be different from call to call. When no token remains in *s1*, a NULL pointer is returned.

For user convenience, all these functions are declared in the optional <**ccs/string.h**> header file.

WARNINGS

Strcmp and *strncmp* are implemented by using the most natural character comparison on the machine. Thus the sign of the value returned when one of the characters has its high-order bit set is not the same in all implementations and should not be relied upon.

Character movement is performed differently in different implementations. Thus overlapping moves may yield surprises.

NAME

string, FONTWIDTH, FONTHEIGHT, smallfont, mediumfont, largefont – draw string in bitmap

SYNOPSIS

```
#include <dmd.h>
#include <font.h>

Point string (ft, s, b, p, f)
Font *ft;
char *s;
Bitmap *b;
Point p;
Code f;

int FONTWIDTH (fnt)
int FONTHEIGHT (fnt)

Font fnt;

Font smallfont;
Font mediumfont;
Font largefont;
```

DESCRIPTION

The *string* function draws the null-terminated string *s* using characters from Font **ft* in Bitmap **b* at Point *p* with function Code *f*. The returned Point value is the location of the first character position following the string *s*. The returned Point can be passed to successive calls to *string* to concatenate strings.

The drawing of the characters is done such that the bounding rectangle of the maximum height character in the font would have its origin at *p*. Therefore, a character drawn on the screen at the point *Drect.origin* will occupy the upper-leftmost character position of the application's window.

The *string* function draws characters as they are in the font. No special action is taken for control characters such as tabs and newlines.

The globals *smallfont*, *mediumfont*, and *largefont* are the names of, not pointers to, the three resident fonts in the 630 MTG.

The *FONTWIDTH* macro returns the width of the space character in the given font. This is only useful if all the characters in the font have the same width. The *FONTHEIGHT* macro returns the height of the given font.

EXAMPLE

The following example demonstrates the use of *string*. The simple subroutine prints "hello world" using the resident font called *largefont* .

```
#include <dmd.h>
#include <font.h>

Point add();
Point string();

hello_world()
{
        Font *f;
        Point p;

        f = &largefont;
        p = add (Drect.origin, Pt(4,4));
        string (f, "hello world", &display, p, F_XOR);
}
```

SEE ALSO

loadfont(1), jstring(3R), structures(3R).

NAME

strtol, atol, atoi – convert string to integer

SYNOPSIS

long strtol (str, ptr, base)
char *str, **ptr;
int base;

long atol (str)
char *str;

int atoi (str)
char *str;

DESCRIPTION

strtol returns (as a long integer) the value represented by the character string pointed to by *str*. The string is scanned up to the first character inconsistent with the base. Leading "white-space" characters [as defined by *isspace* in *ctype*(3L)] are ignored.

If the value of *ptr* is not (char **)NULL, a pointer to the character terminating the scan is returned in the location pointed to by *ptr*. If no integer can be formed, that location is set to *str*, and zero is returned.

If *base* is positive (and not greater than 36), it is used as the base for conversion. After an optional leading sign, leading zeros are ignored, and "0x" or "0X" is ignored if *base* is 16.

If *base* is zero, the string itself determines the base thusly: After an optional leading sign, a leading zero indicates octal conversion, and a leading "0x" or "0X" indicates a hexadecimal conversion. Otherwise, decimal conversion is used.

Truncation from long to int can, of course, take place upon assignment or by an explicit cast.

Atol(str) is equivalent to *strtol(str, (char **)NULL, 10)*.

Atoi(str) is equivalent to *(int) strtol(str, (char **)NULL, 10)*.

SEE ALSO

atof(3L), ctype(3L).

WARNING

Overflow conditions are ignored.

NAME

structures: Word, Code, Point, Rectangle, Bitmap, Texture16, Font, Fontchar, msgbuf, message_list, msqid_ds – 630 MTG Structures

SYNOPSIS

#include <dmd.h>

#include <font.h>

#include <message.h>

DESCRIPTION

In the following summaries, all coordinates are screen or Bitmap coordinates (which are scaled the same) unless specified as window coordinates. *Word , Code , Point , Rectangle , Bitmap* and *Texture16* are included in a program by including **dmd.h**. *Font* and *Fontchar* are defined in **font.h**. *Msgbuf, message_list,* and *msqid_ds* are defined in **message.h**.

Word

```
typedef short Word;
typedef unsigned short UWord;
```

A *Word* is a 16-bit integer and is the unit of storage used in the graphics software.

Code

```
typedef int Code;
```

Code is the functional constant used in all graphical drawing or copying operations. Available Codes are:

```
F_STORE     target  =  source
F_OR        target |=  source
F_XOR       target ^=  source
F_CLR       target &= ~source
```

Point

```
typedef struct Point {
   short  x;      /* x-coordinate */
   short  y;      /* y-coordinate */
} Point;
```

A *Point* is a location in a *Bitmap*, such as the display. The coordinate system has *x* increasing to the right and *y* increasing down.

Rectangle

```
typedef struct Rectangle {
   Point  origin;         /* Upper left corner */
   Point  corner;         /* Lower right corner*/
} Rectangle;
```

A *Rectangle* is a rectangular area in a *Bitmap*. By definition **origin.x<=corner.x** and **origin.y<=corner.y** define the rectangle. By convention, the right (maximum *x*) and bottom (maximum *y*) edges are excluded from the represented rectangle, so abutting rectangles have no points in common. Thus *corner* is the coordinates of the first point beyond

the rectangle. The data on the screen of the 630 MTG is contained in the **Rectangle** {**0, 0, XMAX, YMAX**} where **XMAX=1024** and **YMAX=1024**.

Bitmap

```
typedef struct Bitmap {
   Word      *base;        /* pointer to start of data */
   unsigned short width;   /* width in Words of total data area */
   Rectangle rect;         /* rectangle describing data area */
   char      *_null;       /* unused, must always be zero */
} Bitmap;
```

A *Bitmap* holds a rectangular image stored in contiguous memory starting at *base*. Each *width* words of memory form a scan-line of the image. The *rect* argument defines the coordinate system inside the *Bitmap*. Argument *rect.origin* is the location in the *Bitmap* of the upper-leftmost point in the image and is not necessarily (0,0). Graphical operations performed on a *Bitmap* are clipped to *rect*.

Texture16

```
typedef struct Texture16 {
        Word   bits[16];
} Texture16;
```

A *Texture16* is a 16×16 bit dot pattern. *Texture16*'s are aligned to absolute display positions, so adjacent areas colored with the same *Texture16* align smoothly.

Font and *Fontchar*

```
typedef struct Fontchar
{
   short          x;       /* left edge of character cell in Font.bits */
   unsigned char top;      /* first non-zero scan-line of character image */
   unsigned char bottom;   /* last non-zero scan-line of character image */
   char           left;    /* offset of baseline from x; used for kerning */
   unsigned char width;    /* width of baseline for character image */
} Fontchar;

typedef struct Font
{
   short    n;        /* number of characters in font */
   char     height;   /* height of the Bitmap bits */
   char     ascent;   /* top of Bitmap to baseline of character image */
   long     unused;
   Bitmap   *bits;    /* Bitmap where the characters are stored */
   Fontchar info[1]; /* n+1 character descriptors */
} Font;
```

A *Font* is a character set. For each character in a *Font* there is information stored in a *Fontchar* structure. **Font.info[n]** is a dummy fontchar descriptor used to determine the right edge of the last character in **Font.bits**. The actual character images in the *Font* are stored in a single *Bitmap* pointed to by *bits*. The *Bitmap* contains the bit pattern for each character, arrayed adjacently into a long horizontal strip. The characters in the *Bitmap* must

appear in ASCII order and are aligned on the same baseline. Characters in the *Bitmap* abut exactly, so the width of a character *c* is **Font.info[c+1].x-Font.info[c].x**. When a character is displayed on the screen at a point *p* , the upper left-hand corner of the rectangle enclosing the character image coincides with the point *p*.

msqid_ds, message_list and *msgbuf*

```
typedef struct msgbuf
{
   long        mtype;      /* message type */
   char        mtext[1]    /* text of message */
} msgbuf;

typedef struct message_list
{
   msgbuf *msg;      /* the message in the queue */
   int    size;      /* size of the message */
   struct message_list  *next; /* next message link */
} message_list;

typedef struct msqid_ds
{
   Struct Proc     *cid;          /* creator process id */
   short           msg_qnum;      /* number of messages */
   short           msg_qbytes;    /* max number of bytes */
   Struct Proc     *msg_lspid;    /* last process to send */
   Struct Proc     *msg_lrpid;    /* last process to rcv */
   unsigned long   msg_stime;     /* time of last send */
   unsigned long   msg_rtime;     /* time of last rcv */
   unsigned long   msg_ctime;     /* time of last change */
   message_list    *msg_list;     /* linked message list */
   short           msg_curbytes   /* current # of bytes */
   short           state;         /* remove queue if cid exits? */
   long            name;          /* name of queue (key) */
   struct msqid_ds *next;         /* link to next queue */
} msqid_ds;
```

A *msqid_ds* is a message queue. The messages in the queue are kept in a linked list referenced by the pointer *msg_list*. The actual messages are stored in *msgbuf*s which are pointed to by *message_list*. *Msg_qbytes* is initialized to **MAX_QBYTES** where **MAX_QBYTES=020000** (decimal 8192). The *msqid_ds* is stored in a linked list in the 630 so that the number of message queues is limited only by memory.

FILES

$DMD/include/dmd.h
$DMD/include/font.h
$DMD/include/message.h

NAME

strwidth, jstrwidth – width of character string

SYNOPSIS

#include <dmd.h>
#include <font.h>

int strwidth (f, s)

int jstrwidth (s)

Font *f;
char *s;

DESCRIPTION

The *strwidth* function returns the width in screen coordinates (pixels) of the null-terminated string *s*, interpreted in the Font **f*. The height of a character string is simply *f->height*.

The call, *jstrwidth(s)*, is equivalent to *strwidth (&mediumfont, s)*.

EXAMPLE

The following code fragment places the width of a string that uses the resident font *largefont* into a variable called width.

```
#include <dmd.h>
#include <font.h>

    .
    .

    int width;
    Font *f = &largefont;

    width = strwidth (f, "hello world");

    .
    .
```

SEE ALSO

infont(3L), string(3R), structures(3R).

NAME

swab – swap bytes

SYNOPSIS

void swab (from, to, nbytes)
char *from, *to;
int nbytes;

DESCRIPTION

swab copies *nbytes* bytes pointed to by *from* to the array pointed to by *to*, exchanging adjacent even and odd bytes. *Nbytes* should be even and non-negative. If *nbytes* is odd and positive, *swab* uses *nbytes*–1 instead. If *nbytes* is negative, *swab* does nothing.

NAME

texture – draw Texture16 in Rectangle in Bitmap

SYNOPSIS

#include <dmd.h>

void texture (b, r, t, f)
Bitmap *b;
Rectangle r;
Texture16 *t;
Code f;

Texture16 T_grey, T_lightgrey, T_darkgrey;

Texture16 T_black, T_white, T_background, T_checks;

DESCRIPTION

The *texture* function draws the Texture16 specified by *t* with function Code *f* in the Rectangle *r* in the Bitmap *b*. The Texture16s listed above are predefined.

EXAMPLE

The following program allows doodling with a Texture16.

```
#include <dmd.h>
main()
{
        Rectangle r;
        Point s;
        s.x = 16;
        s.y = 16;
        request (MOUSE);
        for (;;)
        {
                wait(MOUSE);
                r.origin = mouse.xy;
                r.corner = add (r.origin, s);
                if( button3() )
                        break;
                if( button1() )
                        texture (&display, r,
                                &T_grey, F_STORE);
                if( button2() )
                        texture (&display, r,
                                &T_grey, F_CLR);
        }
}
```

SEE ALSO

globals(3R), jtexture(3L), structures(3R).

NAME

tmenuhit – present user with menu and get selection

SYNOPSIS

#include <dmd.h>
#include <menu.h>

Titem *tmenuhit (m, n, flags [, p])
Tmenu *m;
int n;
int flags;
Point p;

void tm_ret()

```
typedef struct Titem
{
        char *text;         /* string for menu */
        struct {
                unsigned short uval;   /* user field */
                unsigned short grey;   /* grey this selection */
        } ufield;
        struct Tmenu *next;   /* ptr to sub-menu */
        Bitmap *icon;         /* ptr to the icons bitmap */
        struct Font *font;    /* font defined for this item */
        void (*dfn)();   /* execute function before sub-menu */
        void (*bfn)();   /* execute function after sub-menu */
        void (*hfn)();   /* execute function on selection */
} Titem;

typedef struct Tmenu
{
        Titem *item;          /* Titem array */
        short prevhit;        /* index to current item */
        short prevtop;        /* index to top item */
        Titem *(*generator)();   /* used if item == 0 */
        short menumap;      /* bit definition of structure */
} Tmenu;

/* bit definitions in menumap */
#define TM_TEXT         0x0001   /* defines text field */
#define TM_UFIELD       0x0002   /* defines ufield field */
#define TM_NEXT         0x0004   /* defines next field */
#define TM_ICON         0x0008   /* defines icon field */
#define TM_FONT         0x0010   /* defines font field */
#define TM_DFN          0x0020   /* defines dfn field */
#define TM_BFN          0x0040   /* defines bfn field */
#define TM_HFN          0x0080   /* defines hfn field */
```

DESCRIPTION

The *tmenuhit* function is an enhanced version of menuhit. It adds such features as expanding menus, use of icons and various fonts within a menu item, greying non-selectable items, and extended control over invocation and specification of the menu facility.

Menu Trees

Menu trees allow the presentation of several menus in a hierarchical format. Each menu is specified by a *Tmenu* structure. Each *Tmenu* structure contains a array of one or more *Titem* structures which specify the menu items. Each item of a menu may then, in turn, point to a submenu. Submenus appear to the right of the parent menu. The presence of a submenu for a menu item is indicated by an arrow icon pointing to the right. Moving the cursor to the arrow icon allows the user to preview the submenu. Sliding further to the right moves the cursor into the submenu and allows the user to make a selection in this menu. Moving the cursor back to the left exits the submenu and moves the cursor back into the parent menu.

Usage

The *tmenuhit* function presents the user with a menu tree specified by the root *Tmenu* pointer *m* and returns a pointer to a *Titem* structure indicating which item was selected. If no item was selected *tmenuhit* returns a 0. The *n* argument is an integer which specifies the mouse button for user interaction: 1, 2, 3 or 0 for all the buttons.. The *flags* argument is a bit vector which indicates various modes of function in *tmenuhit*. These flags include:

(flags & TM_EXPAND)

If true, the menu tree will be expanded (according to the previous selection) down to the lowest leaf on invocation of *tmenuhit*.

(flags & TM_NORET)

If true, *tmenuhit* will not return when a valid selection is made. This feature is useful if a lot of selections are to be made from a large menu.

(flags & TM_STATIC)

If true, *tmenuhit* assumes that no button is depressed when it is called. In this case, the user makes a selection by depressing the button specified when the cursor points to the item desired. If this is false, *tmenuhit* assumes that the button is depressed when called. The user makes a selection by lifting the button when the cursor points to the desired item.

(flags & TM_POINT)

If true, the argument *p* must be present and the origin of the root menu will appear at this point on the display.

The user may define one or more of these flags by or'ing them together within the function call (e.g. (TM_EXPAND|TM_POINT)).

Structure and Functional Description

This section describes the structure fields of the *Tmenu* and the *Titem* structures and the functions of *tmenuhit* they serve.

The **Tmenu** structure defines a menu. It has the following fields:

item This is an array of *Titem* structures which defines each item in the menu. The last item in the array must have its *Titem* text field equal to 0.

prevhit

prevhit is used to store the menu's previous selection. When *tmenuhit* is called the menu is displayed such that, if possible, the mouse cursor will be displayed over the previous selection. This might not be possible if the menu is near the border of the screen. *Prevhit* holds the index from the top of the displayed menu. The *prevhit* value is initialized to 0 and normally does not need to be manipulated by the application program.

prevtop

prevtop is used to store the topmost item displayed in the menu when more than sixteen menu items are defined. The maximum number of items which may be displayed within a menu is sixteen. When there are more than sixteen the menu becomes a **scrolling menu.** In this case, the left portion of the menu contains a scroll bar that is used for scrolling quickly through the menu selections. The vertical size of the scroll bar is an indication of the size of the user's view of the menu (16 items) relative to the number of selections in the entire menu.

There are two ways to scroll through the menu items. The first is to move the mouse cursor to the left side of the menu into the scroll bar area. By moving the mouse cursor up or down within the scroll bar area, the menu items will scroll accordingly. The second method used to scroll through the menu items is to place the mouse cursor on the top or bottom entry of the menu list. The menu will scroll up or down by one item at a time if there are additional items to be displayed in that direction.

Like prevhit, the value of prevtop is initialized to 0 and normally does not need to be manipulated by the application program.

generator

Menu items may be generated dynamically from a program by specifying a generator function in the *Tmenu* structure. If the item field in the *Tmenu* data structure is 0 when a menu is entered, either by calling *tmenuhit* or through the sub-menu mechanism, then the routine specified by *generator* is called with two parameters that are an integer index beginning at 0 and the address of the current *Tmenu*. The generator must return a pointer to a *Titem* structure containing the text for the corresponding menu item. This generator function is called repeatedly with the index increasing by 1 until the generator returns a NULL for the text field in the *Titem* structure, indicating the end of the menu selections.

menumap

In applications where many menus are to be used, the programmer can re-define the *Titem* structure to include only those fields that are actually used. This has the advantage of requiring less data initialization on the part of the programmer. It is done with a bit vector called *menumap* in the *Tmenu* structure. If used, the user defined structure replacing *Titem* must contain the specified member variables in the same order. For example, if one wishes to use only the text, ufield, and next fields of a *Titem* structure, he may define a *Titem* structure with only those fields, and then set the menumap field of the *Tmenu* structure to the value of (TM_TEXT|TM_UFIELD|TMNEXT). Normally, this variable has the value zero when the standard *Titem* structure is used.

Each menu item is defined by a structure of the type *Titem*. The **Titem** structure has the following fields:

text The text field is a pointer to a NULL terminated character string. This is the character string that is displayed within the menu.

A facility provided by *tmenuhit* is that of a spread character. A spread character is any ascii character with the high-order bit set (e.g., an ascii space character defined as a spread character would have the value of '\240'). The spread character acts somewhat like a spring pushing against the adjacent text and borders within a menu entry. The spread character can be placed at the beginning, middle, or end of the string defining the menu entry. If placed at the beginning of the string, the text in the menu item will be right-justified. If placed at the end of the string, the text will be left justified. If placed in the middle of the string, the text on each side of the spread character will be pushed against the corresponding menu border. In each case, the space created by the spread character will be filled in with the ascii character contained in the spread character. For entries without a spread character, the default is to have the text centered.

uval This is an integer to be used for any purpose the user wishes. It is typically used to store a constant that is used by the application to identify *Titem* structures. For instance, this field could be set to a unique value for each menu item in the menu tree. In this way, a switch statement can easily determine the menu item selection regardless of the menu used and the difference in size of different *Titem* structures.

grey If this field is set to 1, the item will be displayed in the menu with a grey background. This item is non-selectable and, if selected, the value *tmenuhit* returns is 0.

next The *next* variable points to another menu structure of type *Tmenu* which defines a submenu for this particular item.

icon The icon field is a pointer to a Bitmap structure that is displayed to the left of the menu item. The size of the bitmap is specified

within the Bitmap structure and can vary from one menu item to the next. The icon can be displayed with or without a text string. However, if icons are to be used without text strings, the value of *text* field cannot be NULL, but must point to a NULL string.

When the icon field is used to display a bitmap, all *Titem* structures for the specific menu are scanned to find the largest bitmap. This is used to determine the vertical spacing of all the menu items within that menu. Smaller bitmaps on other menu entries will be centered within the icon area.

font The font field is a pointer to a font to be used for the text in this menu item. Proportional characters and different point size fonts will be positioned appropriately. If a NULL value is specified, the 630 resident medium font is used.

dfn, bfn, hfn

These three fields may be initialized to point to functions that will be executed by *tmenuhit* before entering a submenu (sliding *down*), after returning from a submenu (sliding *back*), and upon making a selection in the current menu (a *hit*), respectively. Each function is passed, as an argument, the address of the *Titem* structure from which it is called. The hfn function provides an alternative method to using the return value of *tmenuhit.*

In the special case of (flags & TM_NORET), when a selection is made, the submenu the item was in will be erased. Then the hfn function will be called with the selected item. If there is a parent to this menu the bfn and dfn functions will also be called in that order (if they are initialized) after which the menu will be redrawn. This is due to the recursiveness of *tmenuhit.*

If any of the dfn, bfn, or hfn functions execute calls to the function *tmret(), tmenuhit* will ignore the TM_NORET flag and will return the selection made.

EXAMPLE

The following example is a comprehensive example of how one may use *tmenuhit.* The example presents a menu tree when button 3 is depressed and, upon selection of a menu item, it prints the text of that item at the bottom of the screen. The top level menu contains six items each pointing to a submenu. They are *font, test, bttn3, icons, spread,* and *scroll.* The following is a brief explanation of each submenu and the functions it intends to demonstrate.

font The font submenu demonstrates the use of fonts and icons (within the menu item) and the use of the hfn and dfn functions. An hfn function called setfont is executed on selection and sets the font to the selected font. Before the font submenu is displayed, a dfn function called setmark will place a checkmark icon next to the current font in use.

test This submenu contains various sized strings to demonstrate how *tmenuhit* alters its menu item size according to the size of the string.

bttn3 This is a replica of the button3 menu on the 630 MTG. It demonstrates the menu greying capability. This item also demonstrates how one may use *tmenuhit* to achieve the simple functionality of *menuhit* plus the added capability of greying items.

icons This submenu demonstrates the use of icons within menu items.

spread This submenu demonstrates the spread character facility.

scroll This submenu demonstrates the use of generators and presents the scrolling menu.

Another thing to notice is the use of the abbreviated *Titem* structures. There are six different types of *Titem* structures. Each one corresponds to a menu-map vector which defines the fields being used.

```
#include <dmd.h>
#include <font.h>
#include <menu.h>

void setmark(), setfont();
Titem *scrllist();
extern Tmenu menu20, menu21, menu22, menu23, menu24, menu25;
extern Tmenu menu30;

Word strawberry[] = {
0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,
0x0000,0x1B80,0x0000,
0x0000,0x3D80,0x0000,
0x01F8,0x7E80,0x0000,
0x03FE,0x7D80,0x0000,
0x07AF,0xFB80,0x0000,
0x0EF9,0xF780,0x0000,
0x1F57,0xFFFE,0x0000,
0x1FFB,0xFEAB,0x8000,
0x1DAD,0xF77D,0xC000,
0x3FFA,0x65B7,0x4000,
0x3AAF,0x8FDE,0xE000,
0x3FFD,0xFECB,0xA000,
0x3BB7,0x5FEE,0xE000,
0x3EFD,0xFAAB,0xA000,
0x3FAF,0xAFEE,0xE000,
0x3AFA,0xFD6B,0xA000,
0x3FAF,0xD7DF,0xE000,
0x3EFA,0xFED5,0x4000,
0x1FAF,0xABBF,0xC000,
0x1EFA,0xFFAA,0x8000,
0x1FAF,0xD77F,0x8000,
```

```
0x1FFD,0x7ED5,0x0000,
0x0F57,0xD9FE,0x0000,
0x0FFE,0xF754,0x0000,
0x07AB,0xCDFC,0x0000,
0x03FF,0x7EA8,0x0000,
0x00F8,0x3BF0,0x0000,
0x0000,0x1EE0,0x0000,
0x0000,0x0F80,0x0000,
0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,
};

Word help_icon[] = {
    0x0000, 0x0000, 0x0000, 0x0000,
    0xE0E0, 0x6060, 0x7F7E, 0x7DFF,
    0x6FFB, 0x6C7B, 0x6FFE, 0xFFF8,
    0x003C, 0x0000, 0x0000, 0x0000,
};


/* initialize Bitmap structures */
Bitmap bm_strawberry = {
    (Word*) strawberry,   3, 2, 0, 36, 34, 0
};
Bitmap bm_help = {
    (Word*) help_icon,    1, 0, 0, 16, 16, 0
};

/*******************************/
/* definitions for menumap     */
/*******************************/
#define TYPE1       TM_TEXT | TM_NEXT | TM_DFN
#define TYPE2       TM_TEXT
#define TYPE3       TM_TEXT | TM_NEXT
#define TYPE4       TM_TEXT | TM_UFIELD
#define TYPE5       TM_TEXT | TM_ICON
#define TYPE6       TM_TEXT | TM_ICON | TM_FONT | TM_HFN

/*******************************/
/*  define Titem typedef's     */
/*******************************/

/*
 * define a Titem structure with only text, next, and dfn fields
 */
typedef struct Titem01
{
    char *text;     /* string for menu */
    struct Tmenu *next;       /* pointer to sub-menu */
    void (*dfn)();  /* pointer function to execute on submenu */
```

```
} Titem01;

/*
 * define a Titem structure with only text field
 */
typedef struct Titem2
{
    char *text;             /* string for menu */
} Titem2;

/*
 * define a Titem structure with only text and next fields
 */
typedef struct Titem3
{
    char *text;             /* string for menu */
    struct Tmenu *next;     /* pointer to sub-menu */
} Titem3;

/*
 * define a Titem structure with only text and ufield fields
 */
typedef struct Titem4
{
    char *text;             /* string for menu */
    struct {
      unsigned short uval;    /* user field */
      unsigned short grey;    /* flag shows invalid selection */
    } ufield;
} Titem4;


/*
 * define a Titem structure with only text and icon fields
 */
typedef struct Titem5
{
    char *text;     /* string for menu */
    Bitmap *icon;   /* pointer to the icons bitmap */
} Titem5;


/*
 * define a Titem structure with only text, icon,
 * font and hfn fields
 */
typedef struct Titem6
{
    char *text;     /* string for menu */
    Bitmap *icon;   /* pointer to the icons bitmap */
```

```
    void (*hfn)();  /* function to execute on selection */
} Titem6;

/*******************************/
/* initialize Titem structures */
/*******************************/

/*
 * initialize the Titem structure for the main menu
 * has only text, next and dfn fields
 */
Titem01 L1_root[] =
{
    "font",          &menu20,        setmark,
    "test",          &menu21,        0,
    "bttn3",         &menu22,        0,
    "icons",         &menu23,        0,
    "spread",        &menu24,        0,
    "scroll",        &menu25,        0,
     0
};


/*
 * initialize the Titem structure for menu20
 * has only text, icon, font, hfn fields
 */
Titem6 L2_font[] =
{
    "smallfont",     0,          0,     setfont,
    "mediumfont",    0,          0,     setfont,
    "largefont",     0,          0,     setfont,
     0
};

/*
 * initialize the Titem structure for menu21
 * has only text and next fields
 */
Titem3 L2_test[] =
{
    "A long test string so we can see what happens", 0,
    "Short strings", &menu30,
     0
};

/*
 * initialize the Titem structure for menu22
 * has only text and ufield fields
 */
```

```
Titem4 L2_bttn3[] =
{
    "New",          0, 1,
    "Reshape",      0, 1,
    "Top",          0, 1,
    "Bottom",       0, 1,
    "Current",      0, 1,
    "Delete",       0, 1,
    "Exit",         0, 0,
     0
};

/*
 * initialize the Titem structure for menu23
 * has only text and icon fields
 */
Titem5 L2_icons[] =
{
    "strawberry", &bm_strawberry,
    "help", &bm_help,
     0
};


/*
 * initialize the Titem structure for menu24
 * has only text field
 */
Titem2 L2_spread[] =
{
    "left\240",     /* space character with high bit set */
    "\256right",    /* . char with high bit set          */
    "middle",
    "left\337right", /* _ char with high bit set */
    "a very long string",
     0
};

/*
 * initialize the Titem structure for menu30
 * has only text field
 */
Titem2 L3_shorts[] =
{
    "abc",
    "xyz",
    "123",
    "XYZ",
    "ABC",
     0
```

```
/*******************************/
/* initialize Tmenu structures */
/*******************************/
/*
* menu(xy), where x is the level and y is the menu number
*/
Tmenu menu10 = { (Titem *) L1_root, 0, 0, 0, TYPE1 };
Tmenu menu20 = { (Titem *) L2_font, 2, 0, 0, TYPE6 };
Tmenu menu21 = { (Titem *) L2_test, 0, 0, 0, TYPE3 };
Tmenu menu22 = { (Titem *) L2_bttn3, 0, 0, 0, TYPE4 };
Tmenu menu23 = { (Titem *) L2_icons, 0, 0, 0, TYPE5 };
Tmenu menu24 = { (Titem *) L2_spread, 0, 0, 0, TYPE2 };
Tmenu menu25 = { (Titem *) 0, 0, 0, scrllist, 0 };
Tmenu menu30 = { (Titem *) L3_shorts, 0, 0, 0, TYPE2 };

char noselect[] = "no selection";
Font *font;

main()
{
    Titem *ret;

    /* set the font and icon fields in the proper menu */
    L2_font[0].font = &smallfont;
    L2_font[1].font = &mediumfont;
    L2_font[2].font = &largefont;
    L2_font[1].icon = &B_checkmark;
    font = &largefont     /* use the medium font to start with */
    request(MOUSE);
    while(wait(MOUSE)) {
      if (button3()) {
          /* clear the text area for writing strings */
          cursinhibit();
          rectf(&display,
             Rpt(Pt(Drect.origin.x, Drect.corner.y-18),
             Drect.corner), F_CLR);
          cursallow();
          if(ret = tmenuhit (&menu10, 3, TM_EXPAND)) {
             /* write the menu string in text area */
             string(font, ret->text, &display,
                Pt(Drect.origin.x+5, Drect.corner.y-18),
                F_XOR);
          }
          else {  /* no selection was made */
             string(font, noselect, &display,
                Pt(Drect.origin.x+5, Drect.corner.y-18),
                F_XOR);
          }
      }
      else if (button1())
```

```
            exit();
        }
    }

    char digits[10];
    Titem scrlitem;
    char scrlstr[] = "scroll";

    /*
     * generator for scroll menu
     * generate 99 menu items
     */
    Titem *scrllist(i, m)
    int i;
    Tmenu *m;
    {
        int j;

        if (i > 99) {   /* generator stopping condition */
           scrlitem.text = 0;
        }
        else {   /* generate text for items (i.e. "scroll56") */
           scrlitem.text = digits;
           for (j=0; scrlstr[j] != '\0'; j++) digits[j] = scrlstr[j];
           digits[j++] = i/10 + '0';
           digits[j++] = i - (i/10 * 10) + '0';
           digits[j] = '\0';
        }
        return ( &scrlitem );
    }

    /*
     * a dfn function.
     * This is executed before the font submenu is entered
     * place the checkmark by the proper font
     */
    void setmark(mi)
    Titem01 *mi;
    {
        Tmenu *tm;
        Titem6 *tmi;
        int    index;
        int    hit;

        tm = mi->next;
        hit = tm->prevhit + tm->prevtop;
        for (index=0, tmi=(Titem6 *)tm->item; tmi->text; index++,
```

```
        tmi++) { tmi->icon = (index == hit) ? &B_checkmark:0;
    }
}

/*
 * an hfn function
 * this is executed after a selection is made
 */
void setfont(mi)
Titem6 *mi;
{
    /* set the font to the selected font */
    font = mi->font;
}
```

SEE ALSO

menuhit(3L), structures(3R).

WARNINGS

Common uses for user-provided functions in the *Titem* structure include modifying the members of menu data structures such as the *icon* and *grey* fields. The user must be careful that such menu structures are properly initialized.

Whenever a menu is displayed, the screen image obscured by the menu is saved in a bitmap and then later restored when the menu disappears. If the terminal is out of memory and therefore cannot save the screen image, then the menu will be displayed in XOR (exclusive or) mode on top of the existing screen image. Menu items may still be selected in this mode but may be difficult to read. To remedy this problem, memory may be freed up by either deleting or reshaping windows before the menu is displayed.

Because the *tmenuhit* code is recursive, an arbitrary limit to a depth of eight menus is defined to avoid stack overflow.

The user should be careful not to perform screen writes from within the dfn, bfn, or hfn functions. Any writes to the screen from within these functions can corrupt the displayed menu.

NAME

transform, rtransform – window to screen coordinates

SYNOPSIS

```
#include <dmd.h>

Point transform (p)
Point p;

Rectangle rtransform (r)
Rectangle r;
```

DESCRIPTION

The *transform* function returns the screen coordinates of its argument window coordinate Point *p*.

The *rtransform* function returns the screen coordinates of its argument window coordinate Rectangle *r*.

Screen coordinates extend from (0, 0) to (XMAX-1, YMAX-1) and represent the terminal's screen. Window coordinates map (0, 0) to Drect.origin and (XMAX, YMAX) to Drect.corner.

EXAMPLE

The following code will obtain the screen coordinates of the PtCurrent used by the j-routines.

```
#include <dmd.h>
Point p;
Point transform();

p = transform(PtCurrent);
```

The following two routines draw the same line given the same points.

```
#include <dmd.h>
Point transform();

draw1(p,q)
Point p,q;
{
    jsegment (p,q,F_XOR);
}

draw2(p,q)
Point p,q;
{
    segment(&display, transform(p),
            transform(q), F_XOR);
}
```

SEE ALSO

globals(3R), jsegment(3R), muldiv(3L), segment(3R).

NAME

version – return terminal version number

SYNOPSIS

long version ( )

DESCRIPTION

The *version* function returns a hex number which identifies the version of the 630 MTG terminal.

The version number is the equivalent to the ASCII string given as the response to the <**ESC**>**[c** escape sequence. The ASCII string has three fields (f1;f2;f3) defined as follows:

f1	identifies the 630 MTG as a windowing terminal
f2	identifies the terminal as a 630 MTG
f3	identifies the firmware release

The long integer returned by *version* is a hex number rather than an ASCII string for easier parsing by the application program. The hex number has the same three fields in the format 0xf1f2f3, where each field is one byte.

EXAMPLE

For example the hex number:

```
0x080806
```

returned by *version* is equivalent to the ASCII string:

```
8;8;6
```

given in response to the escape sequence <**ESC**>**[c**. This version number corresponds to Release 1.1 of the 630 MTG.

SEE ALSO

dmdversion(1).
630 MTG Terminal User's Guide.

NAME

whathost – determine host connection

SYNOPSIS

int whathost ( )

DESCRIPTION

The *whathost* function returns an identifier for the host to which the process is connected. This value is most useful as an argument to *setupval*. The value -1 is returned if the process is local.

SEE ALSO

local(3R), setupval(3R).

NAME

window: reshape, move, top, bottom, current, delete – window operations

SYNOPSIS

```
#include <dmd.h>

int reshape (r)

int move (p)

void top ( )

void bottom ( )

void current ( )

void delete ( )

Rectangle r;
Point p;
```

DESCRIPTION

The *reshape* function changes the size and/or position of the window in which the process is running. This will change the value of *Drect* and *display*. The *reshape* function will fail and do nothing if the given rectangle lies outside of the screen or is smaller than 32x32. It may also fail because of insufficient memory. On failure, *reshape* will reshape the window back to its original size or to 32x32. When successful, *P->state* is updated, and 1 is returned. Otherwise, 0 is returned.

The *move* function moves the process's window so that *display.origin* lies at the given point. It will fail and do nothing if *p* lies outside of the screen or there isn't enough memory. When successful, *P->state* is updated, and 1 is returned. Otherwise, 0 is returned.

The *top* function brings the window to the top so that it is not obscured by any other window. As its opposite, the *bottom* function puts the window on the bottom so that every window it overlapped will now obscure it.

The *current* function makes the process's window current. This directs the keyboard and mouse input to the process.

The *delete* function deletes the process and its window. This is the same as exit, only the window is also removed. The *delete* function will fail if the window is the last one connected to a host.

SEE ALSO

exit(3R), globals(3R), state(3R).

NAME

bessel: j0, j1, jn, y0, y1, yn – Bessel functions

SYNOPSIS

```
#include <ccs/math.h>

double j0 (x)
double x;

double j1 (x)
double x;

double jn (n, x)
int n;
double x;

double y0 (x)
double x;

double y1 (x)
double x;

double yn (n, x)
int n;
double x;
```

DESCRIPTION

J0 and *j1* return Bessel functions of x of the first kind of orders 0 and 1, respectively. *Jn* returns the Bessel function of x of the first kind of order n.

Y0 and *y1* return Bessel functions of x of the second kind of orders 0 and 1, respectively. *Yn* returns the Bessel function of x of the second kind of order n. The value of x must be positive.

DIAGNOSTICS

Non-positive arguments cause *y0*, *y1* and *yn* to return the value -**HUGE** and to set *errno* to **EDOM**. In addition, a message indicating DOMAIN error is displayed.

Arguments too large in magnitude cause *j0*, *j1*, *y0* and *y1* to return zero and to set *errno* to **ERANGE**. In addition, a message indicating TLOSS error is displayed.

These error-handling procedures may be changed with the function *matherr*(3M).

SEE ALSO

matherr(3M).

NAME

erf, erfc – error function and complementary error function

SYNOPSIS

#include <ccs/math.h>

double erf (x)
double x;

double erfc (x)
double x;

DESCRIPTION

Erf returns the error function of x, defined as $\frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$.

Erfc, which returns 1.0 – *erf(x)*, is provided because of the extreme loss of relative accuracy if *erf(x)* is called for large x and the result subtracted from 1.0 (e.g., for $x = 5$, 12 places are lost).

SEE ALSO

exp(3M).

NAME

exp, log, log10, pow, sqrt – exponential, logarithm, power, square root functions

SYNOPSIS

#include <ccs/math.h>

double exp (x)
double x;

double log (x)
double x;

double log10 (x)
double x;

double pow (x, y)
double x, y;

double sqrt (x)
double x;

DESCRIPTION

Exp returns e^x.

Log returns the natural logarithm of x. The value of x must be positive.

Log10 returns the logarithm base ten of x. The value of x must be positive.

Pow returns x^y. If x is zero, y must be positive. If x is negative, y must be an integer.

Sqrt returns the non-negative square root of x. The value of x may not be negative.

SEE ALSO

hypot(3M), matherr(3M), sinh(3M).

DIAGNOSTICS

Exp returns **HUGE** when the correct value would overflow, or 0 when the correct value would underflow, and sets *errno* to **ERANGE**.

Log and *log10* return **-HUGE** and set *errno* to **EDOM** when x is non-positive. A message indicating DOMAIN error (or SING error when x is 0) is displayed.

Pow returns 0 and sets *errno* to **EDOM** when x is 0 and y is non-positive, or when x is negative and y is not an integer. In these cases a message indicating DOMAIN error is displayed. When the correct value for *pow* would overflow or underflow, *pow* returns $\pm$**HUGE** or 0 respectively, and sets *errno* to **ERANGE.**

Sqrt returns 0 and sets *errno* to **EDOM** when x is negative. A message indicating DOMAIN error is displayed.

These error-handling procedures may be changed with the function *matherr*(3M).

NAME

floor, ceil, fmod, fabs – floor, ceiling, remainder, absolute value functions

SYNOPSIS

```
#include <ccs/math.h>

double floor (x)
double x;

double ceil (x)
double x;

double fmod (x, y)
double x, y;

double fabs (x)
double x;
```

DESCRIPTION

Floor returns the largest integer (as a double-precision number) not greater than x.

Ceil returns the smallest integer not less than x.

Fmod returns the floating-point remainder of the division of x by y: zero if y is zero or if x/y would overflow; otherwise the number f with the same sign as x, such that $x = iy + f$ for some integer i, and $|f| < |y|$.

Fabs returns the absolute value of x, $|x|$.

SEE ALSO

abs(3L).

NAME

gamma – log gamma function

SYNOPSIS

#include <ccs/math.h>

double gamma (x)
double x;
extern int signgam;

DESCRIPTION

Gamma returns $\ln(|\Gamma(x)|)$, where $\Gamma(x)$ is defined as $\int_0^\infty e^{-t}t^{x-1}dt$. The sign of $\Gamma(x)$ is returned in the external integer *signgam*. The argument x may not be a non-positive integer.

The following C program fragment might be used to calculate Γ:

```
if ((y = gamma(x)) > LN_MAXDOUBLE)
        error( );
y = signgam * exp(y);
```

where LN_MAXDOUBLE is the least value that causes *exp*(3M) to return a range error, and is defined in the **<ccs/values.h>** header file.

DIAGNOSTICS

For non-negative integer arguments **HUGE** is returned, and *errno* is set to **EDOM**. A message indicating SING error is displayed.

If the correct value would overflow, *gamma* returns **HUGE** and sets *errno* to **ERANGE**.

These error-handling procedures may be changed with the function *matherr*(3M).

SEE ALSO

exp(3M), matherr(3M), values(5).

NAME

hypot – Euclidean distance function

SYNOPSIS

#include <ccs/math.h>

double hypot (x, y)
double x, y;

DESCRIPTION

Hypot returns

sqrt(x * x + y * y),

taking precautions against unwarranted overflows.

DIAGNOSTICS

When the correct value would overflow, *hypot* returns **HUGE** and sets *errno* to **ERANGE.**

These error-handling procedures may be changed with the function *matherr*(3M).

SEE ALSO

matherr(3M).

NAME

matherr – error-handling function

SYNOPSIS

#include <ccs/math.h>

int matherr (x)
struct exception *x;

DESCRIPTION

Matherr is invoked by functions in the Math Library when errors are detected. Users may define their own procedures for handling errors, by including a function named *matherr* in their programs. *Matherr* must be of the form described above. When an error occurs, a pointer to the exception structure *x* will be passed to the user-supplied *matherr* function. This structure, which is defined in the **<ccs/math.h>** header file, is as follows:

```
struct exception {
        int type;
        char *name;
        double arg1, arg2, retval;
};
```

The element *type* is an integer describing the type of error that has occurred, from the following list of constants (defined in the header file):

DOMAIN	argument domain error
SING	argument singularity
OVERFLOW	overflow range error
UNDERFLOW	underflow range error
TLOSS	total loss of significance
PLOSS	partial loss of significance

The element *name* points to a string containing the name of the function that incurred the error. The variables *arg1* and *arg2* are the arguments with which the function was invoked. *Retval* is set to the default value that will be returned by the function unless the user's *matherr* sets it to a different value.

If the user's *matherr* function returns non-zero, no error message will be printed, and *errno* will not be set.

If *matherr* is not supplied by the user, the default error-handling procedures, described with the math functions involved, will be invoked upon error. These procedures are also summarized in the table below. In every case, *errno* is set to EDOM or ERANGE and the program continues.

EXAMPLE

```
#include <ccs/math.h>

int
matherr(x)
register struct exception *x;
{
        switch (x->type) {
        case DOMAIN:
```

```
        /* change sqrt to return sqrt(-arg1), not 0 */
        if (!strcmp(x->name, "sqrt")) {
              x->retval = sqrt(-x->arg1);
              return (0); /* print message and set
              errno */
        }
    case SING:
        /* all other domain or sing errors, print
           message and abort */
        lprintf("domain error in %s\n", x->name);
        for(;;) sleep(300);
    case PLOSS:
        /* print detailed error message */
        lprintf("loss of significance in %s(%g)=%g\n",
            x->name, x->arg1, x->retval);
        return (1); /* take no other action */
    }
    /* all other errors, execute default procedure */
    return (0);
}
```

DEFAULT ERROR HANDLING PROCEDURES

	Types of Errors					
type	DOMAIN	SING	OVERFLOW	UNDERFLOW	TLOSS	PLOSS
errno	EDOM	EDOM	ERANGE	ERANGE	ERANGE	ERANGE
BESSEL:	–	–	–	–	M, 0	*
y0, y1, yn (arg ≤ 0)	M, –H	–	–	–	–	–
EXP:	–	–	H	0	–	–
LOG, LOG10:						
(arg < 0)	M, –H	–	–	–	–	–
(arg = 0)	–	M, –H	–	–	–	–
POW:	–	–	±H	0	–	–
neg ** non-int 0 ** non-pos	M, 0	–	–	–	–	–
SQRT:	M, 0	–	–	–	–	–
GAMMA:	–	M, H	H	–	–	–
HYPOT:	–	–	H	–	–	–
SINH:	–	–	±H	–	–	–
COSH:	–	–	H	–	–	–
SIN, COS, TAN: –	–	–	–	M, 0	*	
ASIN, ACOS, ATAN2: M, 0	–	–	–	–	–	

ABBREVIATIONS

*	As much as possible of the value is returned.
M	Message is printed (EDOM error).
H	HUGE is returned.
–H	–HUGE is returned.
±H	HUGE or –HUGE is returned.
0	0 is returned.

NAME

sinh, cosh, tanh – hyperbolic functions

SYNOPSIS

```
#include <ccs/math.h>

double sinh (x)
double x;

double cosh (x)
double x;

double tanh (x)
double x;
```

DESCRIPTION

Sinh, *cosh*, and *tanh* return, respectively, the hyberbolic sine, cosine and tangent of their argument.

DIAGNOSTICS

Sinh and *cosh* return **HUGE** (and *sinh* may return **-HUGE** for negative x) when the correct value would overflow and set *errno* to **ERANGE**.

These error-handling procedures may be changed with the function *matherr*(3M).

SEE ALSO

matherr(3M).

NAME

trig: sin, cos, tan, asin, acos, atan, atan2 – trigonometric functions

SYNOPSIS

```
#include <ccs/math.h>

double sin (x)
double x;

double cos (x)
double x;

double tan (x)
double x;

double asin (x)
double x;

double acos (x)
double x;

double atan (x)
double x;

double atan2 (y, x)
double y, x;
```

DESCRIPTION

Sin, *cos* and *tan* return respectively the sine, cosine and tangent of their argument, x, measured in radians.

Asin returns the arcsine of x, in the range $-\pi/2$ to $\pi/2$.

Acos returns the arccosine of x, in the range 0 to π.

Atan returns the arctangent of x, in the range $-\pi/2$ to $\pi/2$.

Atan2 returns the arctangent of y/x, in the range $-\pi$ to π, using the signs of both arguments to determine the quadrant of the return value.

SEE ALSO

matherr(3M).

DIAGNOSTICS

Sin, *cos*, and *tan* lose accuracy when their argument is far from zero. For arguments sufficiently large, these functions return zero when there would otherwise be a complete loss of significance. In this case, a message indicating TLOSS error is displayed. For less extreme arguments causing partial loss of significance, a PLOSS error is generated but no message is displayed. In both cases, *errno* is set to **ERANGE.**

If the magnitude of the argument of *asin* or *acos* is greater than one, or if both arguments of *atan2* are zero, zero is returned and *errno* is set to **EDOM.** In addition, a message indicating DOMAIN error is displayed.

These error-handling procedures may be changed with the function *matherr*(3M).

NAME

font – font file format

DESCRIPTION

A font file is a file containing a description of a Font that can be read by *infont(3R/3L)* or *loadfont(1)* and converted into a Font in the terminal. A font file can be created by using *outfont*.

A font file begins with a structure that is similar to a Font. It looks like the following:

```
struct Fontheader {
    short    n;          /* number of chars in font */
    char     height;     /* height of bitmap */
    char     ascent;     /* top of bitmap to baseline */
    long     unused;     /* in case we think of more stuff */
    Fontchar info[1];    /* n+1 character descriptors */
}
```

The fields in this structure have the same meanings as the ones in the Font structure. There are really $n+1$ Fontchar structures in the info array. The only field that contains valid data in the [n+1]th element is *x*; the leftmost edge of the corresponding cell in the bitmap. Each Fontchar structure starts on a long integer boundary and is padded with null characters to the next long integer boundary and the start of the next Fontchar structure. Therefore, there are two nulls after each of the $n+1$ Fontchars in the file.

Following this in the file is the bitmap image of the font. This is an array holding the bit image of all the characters in the font. It corresponds to *bits->base* in the Font structure. Its size is defined as:

char base[*height*][((*info*[$n+1$].x+31)/32)*4]

The last column of bits used by a font is *info*[$n+1$].x-1. The width is rounded up to the nearest long integer boundary for the bitmap image.

SEE ALSO

infont(3L), loadfont(1), structures(3R).

NAME

ascii – map of ASCII character set

DESCRIPTION

Ascii is a map of the ASCII character set, giving both octal and hexadecimal equivalents of each character, to be printed as needed. It contains:

000 nul	001 soh	002 stx	003 etx	004 eot	005 enq	006 ack	007 bel
010 bs	011 ht	012 nl	013 vt	014 np	015 cr	016 so	017 si
020 dle	021 dc1	022 dc2	023 dc3	024 dc4	025 nak	026 syn	027 etb
030 can	031 em	032 sub	033 esc	034 fs	035 gs	036 rs	037 us
040 sp	041 !	042 "	043 #	044 $	045 %	046 &	047 '
050 (	051)	052 *	053 +	054 ,	055 -	056 .	057 /
060 0	061 1	062 2	063 3	064 4	065 5	066 6	067 7
070 8	071 9	072 :	073 ;	074 <	075 =	076 >	077 ?
100 @	101 A	102 B	103 C	104 D	105 E	106 F	107 G
110 H	111 I	112 J	113 K	114 L	115 M	116 N	117 O
120 P	121 Q	122 R	123 S	124 T	125 U	126 V	127 W
130 X	131 Y	132 Z	133 [	134 \	135]	136 ^	137 _
140 `	141 a	142 b	143 c	144 d	145 e	146 f	147 g
150 h	151 i	152 j	153 k	154 l	155 m	156 n	157 o
160 p	161 q	162 r	163 s	164 t	165 u	166 v	167 w
170 x	171 y	172 z	173 {	174 |	175 }	176 ~	177 del

00 nul	01 soh	02 stx	03 etx	04 eot	05 enq	06 ack	07 bel
08 bs	09 ht	0a nl	0b vt	0c np	0d cr	0e so	0f si
10 dle	11 dc1	12 dc2	13 dc3	14 dc4	15 nak	16 syn	17 etb
18 can	19 em	1a sub	1b esc	1c fs	1d gs	1e rs	1f us
20 sp	21 !	22 "	23 #	24 $	25 %	26 &	27 '
28 (	29)	2a *	2b +	2c ,	2d -	2e .	2f /
30 0	31 1	32 2	33 3	34 4	35 5	36 6	37 7
38 8	39 9	3a :	3b ;	3c <	3d =	3e >	3f ?
40 @	41 A	42 B	43 C	44 D	45 E	46 F	47 G
48 H	49 I	4a J	4b K	4c L	4d M	4e N	4f O
50 P	51 Q	52 R	53 S	54 T	55 U	56 V	57 W
58 X	59 Y	5a Z	5b [	5c \	5d]	5e ^	5f _
60 `	61 a	62 b	63 c	64 d	65 e	66 f	67 g
68 h	69 i	6a j	6b k	6c l	6d m	6e n	6f o
70 p	71 q	72 r	73 s	74 t	75 u	76 v	77 w
78 x	79 y	7a z	7b {	7c |	7d }	7e ~	7f del

NAME

math – math functions and constants

SYNOPSIS

#include <ccs/math.h>

DESCRIPTION

This file contains declarations of all the functions in the Math Library (described in Section 3M), as well as various other functions that return floating-point values.

It defines the structure and constants used by the *matherr*(3M) error-handling mechanisms, including the following constant used as an error-return value:

HUGE	The maximum value of a single-precision floating-point number.

The following mathematical constants are defined for user convenience:

M_E	The base of natural logarithms (e).
M_LOG2E	The base-2 logarithm of e.
M_LOG10E	The base-10 logarithm of e.
M_LN2	The natural logarithm of 2.
M_LN10	The natural logarithm of 10.
M_PI	π, the ratio of the circumference of a circle to its diameter.
M_PI_2	$\pi/2$.
M_PI_4	$\pi/4$.
M_1_PI	$1/\pi$.
M_2_PI	$2/\pi$.
M_2_SQRTPI	$\frac{2}{\sqrt{\pi}}$
M_SQRT2	$\sqrt{2}$
M_SQRT1_2	$\sqrt{\frac{1}{2}}$

For the definitions of various machine-dependent "constants," see *values(5)*.

FILES

$DMD/include/ccs/math.h

SEE ALSO

matherr(3M), values(5).

NAME

values – machine-dependent values

SYNOPSIS

#include <ccs/values.h>

DESCRIPTION

This file contains a set of manifest constants, conditionally defined for particular processor architectures.

The model assumed for integers is two's complement binary, where the sign is represented by the value of the high-order bit.

BITS(*type*)	The number of bits in a specified type (e.g., int).
HIBITS	The value of a short integer with only the high-order bit set (0x8000).
HIBITL	The value of a long integer with only the high-order bit set (0x80000000).
HIBITI	The value of a regular integer with only the high-order bit set (the same as HIBITS).
MAXSHORT	The maximum value of a signed short integer (0x7FFF ≡ 32767).
MAXLONG	The maximum value of a signed long integer (0x7FFFFFFF ≡ 2147483647).
MAXINT	The maximum value of a signed regular integer (the same as MAXSHORT).
MAXFLOAT, LN_MAXFLOAT	The maximum value of a single-precision floating-point number, and its natural logarithm.
MAXDOUBLE, LN_MAXDOUBLE	The maximum value of a double-precision floating-point number, and its natural logarithm.
MINFLOAT, LN_MINFLOAT	The minimum positive value of a single-precision floating-point number, and its natural logarithm.
MINDOUBLE, LN_MINDOUBLE	The minimum positive value of a double-precision floating-point number, and its natural logarithm.
FSIGNIF	The number of significant bits in the mantissa of a single-precision floating-point number.
DSIGNIF	The number of significant bits in the mantissa of a double-precision floating-point number.

FILES

$DMD/include/ccs/values.h

SEE ALSO

math(5).