



999-300-341
Issue 1

630 MTG

Software Development Guide

TRADEMARKS

The following trademarks are used in this manual:

- DEC — Registered trademark of Digital Equipment Corp.
- IBM — Registered trademark of International Business Machines Corporation
- MC68000 — Trademark of Motorola, Inc.
- PDP — Registered trademark of Digital Equipment Corp.
- UNIX — Registered trademark of AT&T
- VAX — Registered trademark of Digital Equipment Corp.
- WE — Registered trademark of AT&T

ORDERING INFORMATION

Additional copies of this document can be ordered by calling

Toll free: 1-800-432-6600 In the U.S.A.

1-800-255-1242 In Canada

Toll: 1-317-352-8557 Worldwide

OR by writing to:

AT&T Customer Information Center
Attn: Customer Service Representative
P.O. Box 19901
Indianapolis, IN 46219

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.

Table of Contents

1. Overview	1-1
Introduction	1-1
User Responsibilities	1-3
Features of the 630 MTG Software Development Package	1-5
Document Organization	1-7

Part 1 Programming the 630 MTG

2. Getting Started	2-1
General	2-1
Some Simple Programs	2-3
3. 630 MTG Operating System Considerations	3-1
Introduction	3-1
The 630 MTG Operating System	3-2
Sharing the CPU	3-4
System and Process Exceptions	3-5
4. Graphics Environment	4-1
Introduction	4-1

Table of Contents

Four Basic Data Types	4-2
Operations, Comparisons and Conversions	4-15
Two Graphical Coordinate Systems	4-21
Graphics Routines	4-26
Example Program - "screen.c"	4-43
5. Application Resources	5-1
Application Resource Management	5-1
The Mouse Resource	5-4
The Keyboard Resource - "kbdchar" and "ringbell"	5-8
The Printer Resource	5-9
Host Communications	5-10
System Services	5-13
Example Programs	5-17
6. User Interface Toolbox	6-1
Introduction	6-1
"menuhit"	6-2
"tmenuhit" - Tree Menus	6-6
The Label Bar	6-15

Message Boxes	6-19
Example Programs	6-21
7. "jx" I/O Interpreter	7-1
Introduction	7-1
How "jx" Works	7-2
Using "jx"	7-3
Functions Available with "jx"	7-4
Example Program	7-5
8. Fonts and the Font Cache	8-1
"Font" and "Fontchar" Structures	8-1
Drawing Characters on the Screen	8-5
Drawing Text Strings - "string", "jstring", and "strwidth"	8-8
Getting New Fonts from the Host - "getfont"	8-9
The 630 MTG Font Cache	8-12
9. Interprocess Communications (Messages)	9-1
Introduction	9-1
What is a Message?	9-2
Creating a Message Queue	9-3

Table of Contents

Sending and Receiving Messages	9-5
Example Program - "messages1.c"	9-8
Message Queue Control	9-11
Example Program - "messages2.c"	9-14
10. Application Caching	10-1
Introduction	10-1
Caching an Application	10-2
Removing Applications from the Cache	10-8
Reshapability of Cached Applications	10-9
Cached Applications and ".text," ".data," and ".bss" Sections	10-11
Writing Shared Text Applications	10-14
Example Programs for Application Caching	10-21
11. Redefining Keyboard Operations	11-1
Introduction	11-1
Redefining Key Clusters	11-2
Redefining the Entire Keyboard	11-6
Demonstrations of Keyboard Redefinition Modes	11-9

Example Programs	11-11
Keyboard Transmittal Codes and Keyboard Layout	11-21

Part 2 Applications

12. Dmdpi Debugger	12-1
Introduction	12-1
Dmdpi User Interface	12-2
Using the Dmdpi Debugger	12-6
Other Dmdpi Features	12-25
“clock.c” Source Code	12-35
13. Jim Text Editor	13-1
Introduction	13-1
The Jim Window	13-2
Jim Operations	13-4
Keyboard Command Summary	13-24
14. Icon Editing	14-1
Introduction	14-1
Bitmaps and Texture16s	14-2

Table of Contents

Using Icon	14-3
15. C Compilation System	15-1
Introduction	15-1
The 630 MTG Compiler	15-3
Other Utilities	15-5
Register Use	15-12
C Language	15-16
16. MC68000 UNIX System Assembler	16-1
Introduction	16-1
Warnings	16-2
Use of the Assembler	16-4
General Syntax Rules	16-5
Segments, Location Counters, and Labels	16-9
Types	16-11
Expressions	16-12
Pseudo-Operations	16-13
Span-Dependent Optimization	16-19
Address Mode Syntax	16-21
Machine Instructions	16-24

Part 3 Appendices and Index

A. Installation and Administration	A-1
General Considerations	A-1
Installation of 630 MTG Software Development Package	A-4
Local Software	A-10
B. 5620 Compatibility	B-1
Co-existence of 5620 DMD and 630 MTG on the Same Host Computer	B-1
Porting 5620 DMD Programs to the 630 MTG . . .	B-3
Index	I-1

List of Figures

Figure 2-1:	Example Program - "hello.c"	2-3
Figure 2-2:	Example Program - "HELLO.c"	2-6
Figure 2-3:	Example Programs "arguments.c"	2-8
Figure 4-1:	Rectangle Edge Exclusion	4-3
Figure 4-2:	Contiguous Block of Memory	4-6
Figure 4-3:	Beginning of "Bitmap" Image in Memory	4-7
Figure 4-4:	Ex_bitmap.width = 2	4-8
Figure 4-5:	Ex_bitmap.width = 3	4-8
Figure 4-6:	"rect" = {{0,0},{48,8}}	4-9
Figure 4-7:	"rect" = {{16,16},{64,24}}	4-10
Figure 4-8:	"rect" = {{10,10},{42,16}}	4-12
Figure 4-9:	"rect" = {{24,5},{44,10}}	4-13
Figure 4-10:	Example Rectangle - "extent" Divided by Two	4-16
Figure 4-11:	Example Rectangle - CenterRectOnPt	4-17
Figure 4-12:	Example Rectangle - "raddp" and "rsubp"	4-18
Figure 4-13:	The Bitmap "physical"	4-22
Figure 4-14:	Global Structures Defining a Window	4-23
Figure 4-15:	Example Program - Bitmap "display"	4-31
Figure 4-16:	Bitmap A	4-34
Figure 4-17:	Bitmap B	4-35
Figure 4-18:	"bitblt" - A Modifying the Image Data of B	4-36
Figure 4-19:	"bitblt" - B Modifying the Image Data of A	4-37
Figure 4-20:	Bitmap A Modified as C	4-38
Figure 4-21:	Bitmap B Modified as D	4-39
Figure 4-22:	"bitblt" - C Modifying Image Data of D	4-40
Figure 4-23:	Source Code for "twist.c"	4-41
Figure 4-24:	Example Program "screen.c"	4-43
Figure 5-1:	Example Program - "mouse.c"	5-17
Figure 5-2:	Example Program "TrackMouse.c"	5-27
Figure 5-3:	Example Program "star.c"	5-29
Figure 5-4:	Example Program "type.c"	5-35
Figure 5-5:	Example Program "vstern1.c"	5-37
Figure 5-1:	Example Program - "mouse.c"	5-17
Figure 5-2:	Example Program "TrackMouse.c"	5-27
Figure 5-3:	Example Program "star.c"	5-29
Figure 5-4:	Example Program "type.c"	5-35
Figure 5-5:	Example Program "vstern1.c"	5-37

List of Figures

Figure 6-1:	Example Program "Menu1.c"	6-21
Figure 6-2:	Example Program "Menu2.c"	6-23
Figure 6-3:	Example Program "Tmenu1.c"	6-27
Figure 6-4:	Example Program "Tmenu2.c"	6-29
Figure 6-5:	Example Program "Tmenu3.c"	6-33
Figure 6-6:	Example Program "Tmenu4.c"	6-37
Figure 6-7:	Example Program "Tmenu5.c"	6-42
Figure 6-8:	Example Program "Tmenu6.c"	6-47
Figure 6-9:	Example Program "Tmenu7.c"	6-52
Figure 6-10:	Example Program "Tmenu8.c"	6-58
Figure 6-11:	Example Program "Tmenu9.c"	6-64
Figure 6-12:	Example Program "label1.c"	6-67
Figure 6-13:	Example Program "label2.c"	6-68
Figure 6-14:	Example Program "label3.c"	6-69
Figure 6-15:	Example Program "msgbox1.c"	6-71
Figure 6-16:	Example Program "msgbox2.c"	6-72
Figure 7-1:	Example Program - "jxmouse"	7-6
Figure 8-1:	Font Bitmap	8-2
Figure 8-2:	Character Image Cell	8-4
Figure 8-3:	Subroutine "drawchar"	8-6
Figure 8-4:	Example Program Using "getfont"	8-10
Figure 8-5:	Example Program - Caching a Downloaded Font	8-13
Figure 9-1:	Example Program - "messages1.c"	9-8
Figure 9-2:	Example Program - "messages2.c"	9-14
Figure 10-1:	Example Program "cache1.c"	10-21
Figure 10-2:	Example Program "cache2.c"	10-22
Figure 10-3:	Example Program "cache3.c"	10-23
Figure 10-4:	Example Program "cache4.c"	10-24
Figure 10-5:	Example Program "cache5.c"	10-25
Figure 10-6:	Example Program "cache6.c"	10-26
Figure 10-7:	Example Program "cache7.c"	10-28
Figure 10-8:	Example Program "cache8.c"	10-30
Figure 10-9:	Example Program "cache9.c"	10-32
Figure 10-10:	Example Program "cache10.c"	10-34
Figure 10-11:	Example Program "cache11.c"	10-36
Figure 10-12:	Example Program "cache12.c"	10-38
Figure 10-13:	Example Program "cache13.c"	10-40
Figure 10-14:	Example Program "cache14.c"	10-42

Figure 10-15:	Example Program "cache15.c"	10-44
Figure 10-16:	Example Program "cache16.c"	10-46
Figure 10-17:	Example Program "cache17.c"	10-48
Figure 11-1:	Example Program "kbd1.c"	11-11
Figure 11-2:	Example Program "kbd2.c"	11-14
Figure 11-3:	Example Program "kbd3.c"	11-18
Figure 11-4:	98-Key Keyboard Transmittal Codes	11-22
Figure 11-5:	98-Key Keyboard Layout	11-25
Figure 11-6:	Keyboard Key Positions	11-26
Figure 13-1:	Sample 630 MTG Screen with Three Jim Frames Open	13-3
Figure 13-2:	Diagnostic Messages	13-18
Figure 14-1:	The icon Editor Display	14-4
Figure 14-2:	The icon Menu	14-5
Figure 16-1:	Assembler Span-Dependent Optimizations	16-20
Figure 16-2:	Effective Address Modes	16-22
Figure 16-3:	MC68000 Instruction Formats	16-25

Chapter 1: Overview

Introduction	1-1
Other 630 MTG Documentation	1-2
User Responsibilities	1-3
Software Package Installation	1-3
User's ".profile"	1-3
Features of the 630 MTG Software Development Package	1-5
Applications for All Users	1-5
Applications for Programmers	1-6
Document Organization	1-7

Introduction

The AT&T 630 MTG Software Development Package provides tools that enable a C language programmer to write, compile, execute and debug application programs on and for the AT&T 630 Multitasking Terminal with Graphics (MTG). This document provides detailed information on using the tools provided in the 630 MTG Software Development Package and writing custom C language programs for the 630 MTG.

This document is intended primarily for experienced C language programmers; however, several sections are useful to all 630 MTG users. The 630 MTG Software Development Package must be installed on your host computer before you attempt to write or execute application programs on the 630 MTG.

This document includes information on the following:

- Other 630 MTG documentation
- Writing custom 630 MTG application programs
- Using the 630 MTG **dmdpi** C language debugger
- **jim** text editor
- Icon editing
- C Compilation System.

The appendices of this document include information on:

- Installation and administration of the 630 MTG Software Development Package
- Compatibility between the 630 MTG and the AT&T 5620 Dot-Mapped Display terminals.

Other 630 MTG Documentation

Other documents that provide information on the 630 MTG and the 630 MTG Software Development Package are:

- *630 MTG Terminal User's Guide* (document number 999-300-375) provides general user information. You should consult this guide for installation and operation details.
- *630 MTG Software Reference Manual* (document number 999-300-340) provides manual pages on the 630 MTG applications and library functions. You should refer to the manual pages for specific usage information on commands and application programs associated with the 630 MTG Software Development Package.
- *630 MTG Software Development Package Release Notes* (document number 999-300-342) provides a listing of all files associated with the 630 MTG Software Development Package and any known software exceptions.
- *630 MTG Service Manual* (document number 582-630-030) provides installation, setup, diagnostics, disassembly, adjustment, and cabling information.

User Responsibilities

Software Package Installation

The 630 MTG Software Development Package must be installed on your 630 MTG host computer. Appendix A, "Installation and Administration," gives all the necessary information for a system administrator to install and set up the 630 MTG Software Development Package. Dependencies, storage requirements, and installation information for the 630 MTG Software Development Package are also included.

User's ".profile"

To use the 630 MTG Software Development Package, make the following changes to your *.profile*:

- Set the shell variable **\$DMD** to the root of the 630 MTG Software Development Package tree. This is normally the directory **/usr/opt/630**. If this directory does not exist on your host computer, get the correct directory information from the person who installed the 630 MTG Software Development Package.
- The directory **\$DMD/bin** must be added to the **\$PATH** variable.
- The shell variable **\$TERM** should be set to **630**. This is not required to use the 630 MTG host software, but it is needed by other UNIX* System V applications (such as *vi*) which use the UNIX **terminfo** facility. See the *630 MTG Terminal User's Guide* for more information on **terminfo**.

Note: For convenience, a copy of the **terminfo** information given in the *630 MTG Terminal User's Guide* is included in the 630 MTG Software Development Package under **\$DMD/terminfo**.

* Registered trademark of AT&T

User Responsibilities

- The shell variables **DMD**, **PATH**, and **TERM** must all be exported.

The following is an excerpt from a `.profile` which will set up your environment for the 630 MTG and the 630 MTG Software Development Package.

```
stty tabs ixon  
DMD=/usr/opt/630  
PATH=$PATH:$DMD/bin  
TERM=630  
export DMD PATH TERM
```

Features of the 630 MTG Software Development Package

The 630 MTG Software Development Package provides a program development and execution environment. In addition, a small number of downloadable applications and firmware support programs are included. This package must be installed on the host computer in order to run any programs written for the 630 MTG.

The following list gives a brief description of the application programs that are included in the 630 MTG Software Development Package. (Programs that are applicable to all 630 MTG users are listed first, followed by programs that are applicable only to 630 MTG programmers.)

Note: The most complete source of specific information on each program is the individual manual page located in the *630 MTG Software Reference Manual*.

Applications for All Users

- **dmdversion** — Reports the version numbers of the firmware in the 630 MTG and the 630 MTG Software Development Package on the host computer.
- **dmddemo** — Provides several simple graphics "demos" which demonstrate the graphics capabilities of the 630 MTG.
- **dmdcat** — Sends specified files to a printer connected to the Auxiliary Port of the 630 MTG. This command sends the concatenation of files specified on the command line or the standard input (if no files are specified).
- **loadfont** — Allows the user to load and remove fonts from the 630 MTG's font cache. Fonts are loaded from files on the host computer. A number of fonts are provided in the 630 MTG Software Development Package in the directory **\$DMD/termfonts**.
- **jim** — Provides the mouse-driven, visual, text editor for the 630 MTG. (See the "Jim Text Editor" Chapter in this document for more information.)
- **dmdman** — Prints the on-line manual pages for command(s) given on the command line. (Manual pages are not available on the AT&T 3B2 host computers.)

Applications for Programmers

- **C Compilation System (CCS)** — Provides a C language cross compiler for compiling programs to be downloaded into the 630 MTG. The CCS also contains a number of utility programs to read and manipulate object files. (See the "C Compilation System" Chapter of this document for more information.)
- **dmdcc** — Compiles C language programs in a manner similar to the UNIX System V **cc** command. **dmdcc** is the interface to the C cross compiler. **include** files for compiling 630 MTG programs as well as host libraries are also a part of the 630 MTG Software Development Package.
- **Dmdd** and **jx** - Download application programs for execution in the 630 MTG. The **jx** application, a superset of **dmdld**, downloads the application and then remains active, simulating standard I/O library functions (such as file I/O) upon request from the application executing in the terminal.
- **dmdpi** — Provides an interactive, multi-window, mouse-driven debugger for C language programs downloaded into the 630 MTG. (See the "Dmdpi Debugger" Chapter for information on how to use the dmdpi debugger.)
- **dmdmemory** — Presents a graphical representation of memory usage in the 630 MTG. A user will be able to monitor memory, modify allocation limits, and observe memory usage of a particular process.
- **icon** — Provides an interactive icon and picture drawing program. (See the "Icon Editing" Chapter for more information.)

Document Organization

This document is organized as follows:

- Chapter 1 — "Overview" provides introductory information for this document and the 630 MTG Software Development Package.
- Part 1** "Programming the 630 MTG" provides a tutorial on writing C language programs for the 630 MTG.
- Chapter 2 — "Getting Started" provides introductory information on programming the 630 MTG.
 - Chapter 3 — "630 MTG Operating System Considerations" provides information on the 630 MTG program execution environment.
 - Chapter 4 — "Graphics Environment" describes the different features that are used to manipulate and display images on the 630 MTG screen.
 - Chapter 5 — "Application Resources" describes the various 630 MTG application resources, such as the mouse, keyboard, printer, host communications, etc.
 - Chapter 6 — "User Interface Toolbox" describes the various "tools" that provide the user interface for an application program.
 - Chapter 7 — "jx I/O Interpreter" describes the standard 630 MTG I/O interpreter.
 - Chapter 8 — "Fonts and the Font Cache" provides information on "Font" and "Fontchar" data structures and the Font Cache for defining and displaying characters on the 630 MTG screen.
 - Chapter 9 — "Interprocess Communications (Messages)" describes the internal mechanism for communicating between applications running on the 630 MTG.
 - Chapter 10 — "Application Caching" describes the facilities for caching applications that have been downloaded to the 630 MTG.

Document Organization

- Chapter 11 — "Redefining Keyboard Operations" describes how to redefine specific key clusters or the entire keyboard to suit the specific needs of an application.

Part 2 "Applications" provides user information on features provided by the 630 MTG Software Development Package.

- Chapter 12 — "Dmdpi Debugger" provides how-to-use information on the C language dmdpi debugger.
- Chapter 13 — "Jim Text Editor" provides user information on the 630 MTG visual text editor.
- Chapter 14 — "Icon Editing" explains how to create icons using the **icon** program.
- Chapter 15 — "C Compilation System" provides a tutorial on the components of the CCS.
- Chapter 16 — "MC68000 UNIX System Assembler" provides a reference manual for MC68AS, the assembler language interpreted by the **mc68as** assembler.

Part 3 "Appendices and Index"

- Appendix A — "Installation and Administration" provides information to install and maintain the 630 MTG Software Development Package.
- Appendix B — "5620 Compatibility" provides information about the co-existence of 630 MTG and 5620 DMD terminals on the same host computer. Information is also included on porting 5620 DMD programs to the 630 MTG.

Chapter 2: Getting Started

General	2-1
Example Programs	2-1
What You Need to Know	2-1
Materials Needed	2-2
Porting 5620 Programs	2-2
Some Simple Programs	2-3
Example Program - "hello.c"	2-3
Compiling hello.c - "dmdcc"	2-3
Download and Execution - dmdld	2-4
Notes on "hello.c"	2-4
Example Program - "HELLO.c"	2-5
Notes on "HELLO.c"	2-7
Example Program - "arguments.c" ("argc" and "argv")	2-8
Notes on "arguments.c"	2-9

General

This part describes how to develop application programs for the 630 MTG. Although written for programmers who are not familiar with the 630 MTG, this programming section provides detailed reference information that is needed by all 630 MTG programmers, regardless of their experience.

Example Programs

The best way to learn about programming the 630 MTG is through experience. Several example programs have been provided in each chapter to demonstrate important programming features. These programs will allow you to focus attention on the use of specific library routines as they are discussed. The source code for each of the example programs has been provided with the 630 MTG Software Development Package and may be found in the directory *\$DMD/examples*. A printout of the source code listings for the example programs is also included in each chapter. (Short listings are included in the text while longer listings are included in the "Example Programs" section at the end of each chapter.)

What You Need to Know

This part assumes that you are familiar with the following:

- UNIX System V operating system
- C programming language
- **Layers** windowing environment
- A text editor that will work with the 630 MTG such as **jim**
- *630 MTG Terminal User's Guide*.

The *630 MTG Terminal User's Guide* describes how to install, operate, and care for the 630 MTG. You should be particularly familiar with the Chapters "Mouse and Menu Operation," "Windowing Operations," and "Windowproc," in the *630 MTG Terminal User's Guide*.

Materials Needed

In order to get the most from this programming section, you should have a 630 MTG terminal connected to a host computer running the UNIX System V operating system with the 630 MTG Software Development Package installed. See the "Installation of 630 MTG Software Development Package" section in Appendix A for instructions on how to install the 630 MTG Software Development Package. You should also have a copy of the *630 MTG Software Reference Manual* that provides a detailed explanation of the tools and library routines available in the 630 MTG Software Development Package.

Porting 5620 Programs

Programs written for the 5620 Dot-Mapped Display (the forerunner of the 630) can be ported to the 630 MTG with minor changes and recompilation. This information on porting 5620 DMD programs to the 630 MTG is included in Appendix B of this document.

Some Simple Programs

This section will lead the first-time 630 MTG programmer through the steps of creating, compiling, downloading, and executing a few simple 630 MTG applications. (The words "application" and "program" are used interchangeably in this document to refer to any program written for the 630 MTG.)

Example Program - "hello.c"

The best way to learn how to program the 630 MTG is to start simple and build on that. For example, start with a simple program that prints the words "hello, world". The program to accomplish this is shown in Figure 2-1.

```
#include <dmd.h>

main()
{
    lprintf("hello, world");
    for(;;) wait (CPU);
}
```

Figure 2-1: Example Program - "hello.c"

The source code for this example program can be found in the file named **hello.c** in the directory *\$DMD/examples/GettingStarted*. Copying this file into your directory will save you the time of creating it yourself.

Compiling hello.c - "dmdcc"

The C compiler in the 630 MTG Software Development Package is called **dmdcc** and is analogous to the UNIX System C compiler **cc**. Programs written for the 630 MTG must be compiled using **dmdcc**. The following command will compile **hello.c** and produce a 630 MTG executable object called **dmda.out** (the host system's prompt is not shown):

```
dmdcc hello.c
```

For more details concerning the 630 MTG C compiler, see the manual page for **dmdcc** in the *630 MTG Software Reference Manual*.

Download and Execution - `dmdld`

You are now ready to transfer the executable object `dmda.out` from your host computer to the 630 MTG's memory. This transfer is called a "download." The 630 MTG "downloader" is called `dmdld`. The following command will download `dmda.out`:

```
dmdld dmda.out
```

When the download begins, your window will be cleared and will start filling from the bottom with inverse video. You will also notice that when the mouse cursor is in the window, it changes to a coffee cup. The download is complete when the inverse video reaches the top of the window. Execution of the program begins immediately after the download is finished. As expected, the words "hello, world" are displayed. The program can be terminated by deleting the window.

Notes on "hello.c"

All applications that run on the 630 MTG must include the file "`dmd.h`". This file is in the directory `$DMD/include` and you need not make your own local copy since `dmdcc` knows where to find it. `dmd.h` declares several primitive data types needed by applications that run on the 630 MTG.

Printing Text - "lprintf"

The 630 MTG Software Development Package has a number of library routines that allow you to print text on the screen. In the example program `hello.c`, the library routine `lprintf` is used to print the string "hello, world". `lprintf` is syntactically equivalent to the UNIX System's standard I/O `printf` function. It is called by an application when it wants to display text locally in its window. `lprintf` calls another 630 MTG library routine called `lputchar` which does the actual printing of individual characters. `lputchar` is syntactically equivalent to the UNIX standard I/O `putchar` function. Further details on the library routines `lprintf` and `lputchar` can be found in the *630 MTG Software Reference Manual* on the manual pages `PRINTF(3L)` and `LPUTCHAR(3L)`.

Sharing the CPU - "wait(CPU)"

In the 630 MTG execution environment, the central processing unit (CPU) is a shared resource. When an application starts execution in the 630 MTG, it becomes the "owner" of the CPU and as long as it retains this "ownership," no other application in the 630 MTG will be allowed to run. The function call **wait(CPU)** lets your application share the CPU with other applications running in the terminal. When **wait** is called with the argument **CPU**, ownership of the CPU is relinquished until all other applications in the 630 MTG have had a chance to run. It is very important to share the CPU since not doing so will prevent 630 MTG system processes from running. Further details about sharing the CPU will be given in the next chapter.

Example Program - "HELLO.c"

The second example program (see Figure 2-2) gets a little fancier with the way it says "hello." It prints the word "Hello" alongside a graphical representation of the world.

Some Simple Programs

```
#include <dmd.h>
#include "world.h"

/* Library Routines and associated manual page. */
void bitblt();      /* BITBLT(3R)          */
void lprintf();    /* PRINTF(3L)          */
Point sPtCurrent(); /* MOVETO(3L)          */
int wait();        /* RESOURCES(3R)       */

main()
{
    /*
    ** See manual page STRUCTURES(3R) for the
    ** Point data type.
    */
    Point savept;

    /*
    ** Use "lprintf" to move the "current screen point"
    ** and set a position at which to display the
    ** image of the world.
    */
    lprintf("\n          ");
    savept = sPtCurrent();

    lprintf("\n Hello,          !");

    /*
    ** Display the image of the world.
    */
    bitblt(&world, world.rect, &display, savept, F_XOR);

    /*
    ** Share the CPU with other applications
    ** and 630 MTG system processes.
    */
    for (;;) wait(CPU);
}
```

Figure 2-2: Example Program - "HELLO.c"

To compile this example, you will need to copy the source file **HELLO.c** and include file **world.h** into your directory from *\$DMD/examples/GettingStarted*. As before, you can compile and download the program using the commands:

```
dmdcc HELLO.c  
dmdld dmda.out
```

Notes on "HELLO.c"

"lprintf" and the Current Screen Point

For every application running in the 630 MTG, there is a point in its window designated as the "current screen point". When an application begins execution, the "current screen point" is set to the upper left-hand corner of the window. **lprintf** uses the "current screen point" to determine where it should start printing a text string. It also takes care of updating the "current screen point" so that the next text string printed will be concatenated with the last. The coordinates of the "current screen point" can be determined by calling the routine **sPtCurrent**. See the Chapter "Graphics Environment" for more details on the "current screen point".

HELLO.c uses **lprintf** to move the "current screen point" to a location where the world image will be displayed. This is the purpose of the first call to **lprintf**. **sPtCurrent** is then called to save the "current screen point" in the **savept** variable (the "Point" data type will be discussed in detail in the chapter on graphics). The second call to **lprintf** prints the string "Hello" and the exclamation point.

Display the Image of the World

The routine **bitblt** is then called to display the image of the world. Basically, **bitblt** allows a rectangular image stored in one memory location to be copied to a specific location within a second rectangular image. In **HELLO.c**, the rectangular image of the world defined in the include file **world.h** is copied to **savept** in the application's window. Much more will be said about **bitblt** in the Chapter "Graphics Environment."

Example Program - "arguments.c" ("argc" and "argv")

The final example program of this chapter (see Figure 2-3 for source code) demonstrates how an application can use **argc** and **argv**.

```
#include <dmd.h>

/* Library Routines and associated manual page. */
void lprintf();      /* PRINTF(3L)          */
int wait();          /* RESOURCES(3R)       */

main(argc, argv)
int argc;
char **argv;
{
    int i;
    /*
    ** Print command line arguments.
    */
    for (i=0; argc; argc--, i++)
        lprintf("\n argument %d = %s", i, argv[i]);

    /*
    ** Share the CPU with other applications
    ** and 630 MTG system processes.
    */
    for (;;) wait(CPU);
}
```

Figure 2-3: Example Programs "arguments.c"

If your application needs to have command line arguments, they can be specified on the **dmdld** command line as follows:

```
dmdld dmda.out arg1 arg2 ...
```

The arguments are then available in the standard way through `argc` and `argv`. Copy `arguments.c` into your directory from `$DMD/examples/GettingStarted`. Compile and download the application as follows:

```
dmdcc arguments.c  
dmdld dmda.out arg1 arg2 arg3 "Hello there"
```

Notes on “arguments.c”

Note that `argv[0]` contains the name of the downloaded object `dmda.out`. All other arguments are printed as specified on the `dmdld` command line.

Chapter 3: 630 MTG Operating System Considerations

Introduction	3-1
The 630 MTG Operating System	3-2
Processes	3-2
The Process Structure and the Global Variable "P"	3-2
Process States and Scheduling	3-3
Sharing the CPU	3-4
System and Process Exceptions	3-5
Process Exceptions	3-5
System Exceptions	3-5

Introduction

This chapter presents some fundamental considerations about the 630 MTG operating system and its execution environment that will affect how efficiently your applications run in this environment.

The 630 MTG Operating System

The basic structure of the 630 MTG operating system is a set of independent processes scheduled to run in a round-robin fashion.

Processes

A process is the execution of a program along with all the operating system overhead needed to have that program execute. There are two types of processes that run in the 630 MTG: system processes and application processes.

System processes do not have associated windows on the screen and are always scheduled to be run by the operating system. There are three basic system processes in the 630 MTG operating system:

1. Control process - handles global mouse interaction and window manipulation
2. Keyboard process - handles the translation of raw codes received from the 630 MTG keyboard
3. I/O process - handles distribution of host input and output data.

An application process has an associated window and the initial scheduling of the application is controlled by the user. Examples of application processes include: **Windowproc**, **Setup**, **Pfedit**, and downloaded applications.

The Process Structure and the Global Variable "P"

Associated with each process is a data structure called **Proc**. This data structure, most commonly referred to as the "process structure", contains many entries and serves to uniquely identify each active process. A number of the entries in the process structure are particularly useful for application programs and will be discussed in later chapters as needed. The process structures associated with all currently active processes are linked together in a list called the "process list". The global variable "P" points to the process structure associated with the process currently in the "run state".

Process States and Scheduling

Every process in the 630 MTG is in one of three states: running, ready, or waiting. Since the 630 MTG has only one CPU there can only be one process in the run state at a time. All others are either ready to run or waiting for a particular resource. Scheduling of processes in the 630 MTG is non-preemptive. Thus, it is the responsibility of the process in the run state to service its current needs and then relinquish ownership of the CPU so that the next process in the "process list" can run. The switch from one process to the next is handled by the routine **wait**.

Sharing the CPU

A common programming error made in 630 MTG applications is the failure to share the CPU. As previously stated, scheduling of processes in the 630 MTG is non-preemptive. Therefore, if you forget to initiate a process switch by calling the **wait** routine from your application, you will inhibit all other processes in the terminal from running, including the system processes. This has the undesired effect of locking up your terminal. There are two ways to recover from this error: either execute the 630 MTG Selftest, or turn off your terminal. Refer to the *630 MTG Terminal User's Guide* for information on doing a User-Initiated Selftest.

System and Process Exceptions

An exception occurs in the 630 MTG when the CPU tries to execute an illegal operation; for example, trying to divide by zero or accessing a memory location that is not defined in the 630 MTG address space. There are two categories of exceptions that can occur in the 630 MTG: "system exceptions" and "process exceptions". System exceptions are generated by system processes and process exceptions are generated by application processes.

Process Exceptions

When a process exception occurs, a message will be displayed at the bottom of the window of the offending application process. The user will be told to type any key to restart the window. Pressing any key will reset the window as if it had just been created. In most cases everything will be fine after a process exception is cleared, but this may not always be the case.

Warning: Just because the process exception can be cleared by typing a key on the keyboard is no guarantee that everything is operating properly. The application process may have corrupted memory being used by other processes, since a process can write anywhere in the 630 MTG's address space.

If you are in doubt about the sanity of your terminal, you should turn it off and start over again.

System Exceptions

System exceptions are more severe than process exceptions. System exceptions are usually the result of bugs in the application process corrupting memory used by system processes. When a system exception occurs, a message line will appear on the bottom of the screen describing the exception. In order to restart the terminal, the user is told to type any key. Pressing a key will cause the terminal to be reset, as if the power was turned off and then on again.

Chapter 4: Graphics Environment

Introduction	4-1
Four Basic Data Types	4-2
"Point"	4-2
"Rectangle"	4-2
"Word"	4-4
"Bitmap"	4-5
"Bitmap" Illustrations	4-6
Include File "world.h"	4-13
Operations, Comparisons and Conversions	4-15
Operations on "Points"	4-15
"Point" Comparison	4-16
"Rectangle" Operations	4-17
"Rectangle" Comparison	4-18
Inclusion Operations	4-19
Data Type Conversions	4-19
Two Graphical Coordinate Systems	4-21
Global Structures Describing the Two Coordinate Systems	4-21
Current Point	4-23
Current Point in the Screen Coordinate System	4-24
Current Point in the Window Coordinate System	4-24
Coordinate System Transformations	4-25
Graphics Routines	4-26
Function Codes	4-26
Drawing Routines	4-27
Screen Coordinate Drawing Routines	4-27
Window Coordinate Routines	4-29
Textures	4-30
"bitblt"	4-32
"bitblt" Illustrations	4-33

Chapter 4: Graphics Environment

Using "bitblt" - "twist.c" 4-41

Example Program - "screen.c" 4-43

Introduction

The 630 MTG is a "raster graphics" display terminal. The screen consists of a 1024 by 1024 array of pixels, with each pixel represented by a single bit in memory that is either "on" or "off". The entire screen image is represented by a 128K byte "refresh buffer".

In order to manipulate and display images on the screen, you will need a basic understanding of the following items that are described in this chapter:

- Different graphical data types
- Global structures
- Coordinate systems
- Operations on data types
- Data type transformations
- Drawing routines.

Four Basic Data Types

The Graphical Data Types presented in this section are defined in the include file `dmd.h`. This file must be included by all applications that run on the 630 MTG.

“Point”

The **Point** data type is used to specify the x- and y-coordinates of a pixel on the screen or a bit in memory. The **Point** data structure consists of two 16-bit integers and is defined as follows:

```
typedef struct Point {
    short    x;
    short    y;
} Point;
```

By convention, the x-coordinate increases from left to right and the y-coordinate increases from top to bottom on the screen.

“Rectangle”

The **Rectangle** data type is specified by two **Points** and is used to define a rectangular region on the screen or in memory. The **Rectangle** data type is defined as follows:

```
typedef struct Rectangle {
    Point origin;
    Point corner;
} Rectangle;
```

The **origin** specifies the upper left corner (minimum x- and y-coordinates) of the **Rectangle**, and **corner** specifies the lower right corner (maximum x- and y-coordinates). Stated algebraically:

$$\text{origin.x} \leq \text{corner.x} \quad \text{and} \quad \text{origin.y} \leq \text{corner.y}$$

By convention, the right (maximum x-coordinate) and bottom (maximum y-coordinate) edges of a **Rectangle** are excluded. This is so that abutting **Rectangles** have no points in common. See Figure 4-1 and the explanation that follows for more details on rectangle edge exclusion.

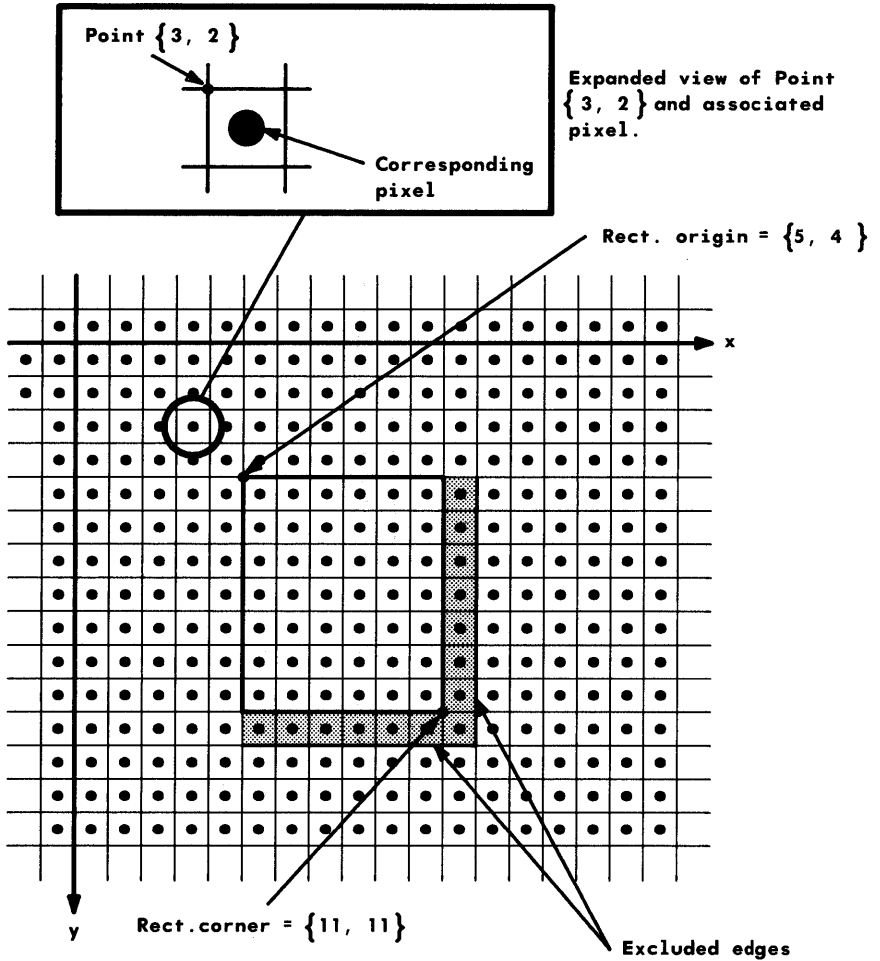


Figure 4-1: Rectangle Edge Exclusion

Four Basic Data Types

Figure 4-1 illustrates the relationship between **pixels**, **Points** and **Rectangles**. An xy-coordinate system is shown with "x" increasing to the right and "y" increasing downwards. Each square in the grid represents a single **pixel** with its center indicated by the dot. The upper left corner of the square designates the **Point** that corresponds to the **pixel**. In this way, a **Point** can be thought of as a common corner of four adjacent **pixels** but only designates the **pixel** immediately below it and to the right. See the expanded view of Point {3, 2} in Figure 4-1.

A **Rectangle** can be thought of as an imaginary box bounded, in the x- and y-direction, by two **Points** (not pixels). Only those **pixels** that have their centers within the box are included in the **Rectangle**. Notice that the **pixels** associated with the right and bottom edges of the **Rectangle** are not enclosed by the box. This is how the right and bottom edges of the **Rectangle** are excluded.

“Word”

The 630 MTG employs a Motorola MC68000* microprocessor as its Central Processing Unit (CPU). The MC68000 fetches data from memory in 16-bit increments. Therefore, the basic quantum of memory used by the 630 MTG's graphics software is a 16-bit integer called a **Word** which is defined as follows:

```
typedef short Word;
```

The number of bits in a **Word** is defined by the constant **WORDSIZE** in the include file **dmd.h**.

* Trademark of Motorola, Inc.

“Bitmap”

A **Bitmap** defines a storage area in memory for a rectangular image. The **Bitmap** data structure is defined as follows:

```
typedef struct Bitmap {  
    Word *base;  
    unsigned short width;  
    Rectangle rect;  
    char *_null;  
} Bitmap;
```

- base** This is a pointer to the first **Word** of a contiguous block of memory. This block of memory contains the **Bitmap** image representation.
- width** This segments the block of memory referenced by **base** into scan lines. **width** is the number of **Words** in a scan line.
- rect** This establishes a coordinate system in which the **Bitmap** image resides and defines the **Bitmap** image boundary. All bits or **Words** outside of this boundary are ignored by graphical operations.
- rect.origin** This is the coordinate of the upper left **Point** in the **Bitmap** image. This **Point** always corresponds to a bit in memory that resides within the **Word** referenced by **base**.
- _null** This must be set to (char *)0.

“Bitmap” Illustrations

This section illustrates the elements of the **Bitmap** data structure just defined. Consider the segment of contiguous memory shown in Figure 4-2. Each rectangle in the illustration represents a **Word** and has been numbered to establish some order. We will assume that somewhere in this segment of memory is the representation of a **Bitmap** image called **Ex_bitmap**.

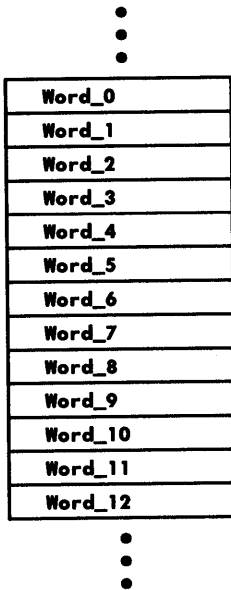


Figure 4-2: Contiguous Block of Memory

In order to be specific about where the **Bitmap** image resides in memory, we must specify **Ex_bitmap.base**. **Ex_bitmap.base** specifies the first **Word** of the **Bitmap** image. See Figure 4-3.

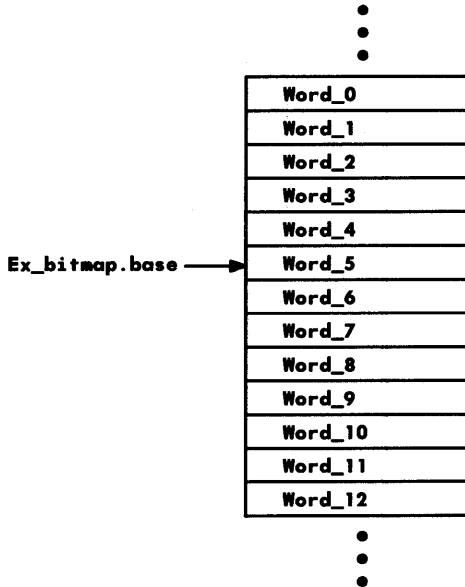


Figure 4-3: Beginning of "Bitmap" Image in Memory

Four Basic Data Types

Consider two different specifications for `Ex_bitmap.width`. Figure 4-4 shows how the **Bitmap** image memory would be viewed for **width** equal to 2 and Figure 4-5 for **width** equal to 3. Recall that **width** segments the **Bitmap** image memory into scan lines.

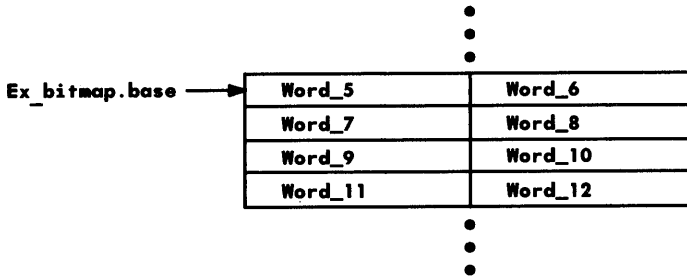


Figure 4-4: `Ex_bitmap.width = 2`

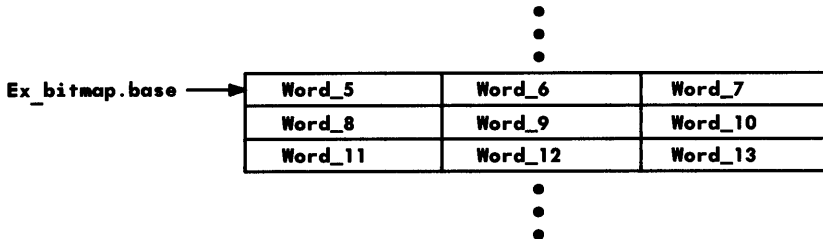


Figure 4-5: `Ex_bitmap.width = 3`

Finally, consider various specifications of `Ex_bitmap.rect`.

A. Consider the following `Bitmap` specification:

```
Bitmap Ex_bitmap={
    &Word_5;
    3,
    {{0,0} , {48,8}},
    (char *)0,
};
```

`Ex_bitmap.rect` specifies a boundary that is 48 bits wide and 8 bits high. Figure 4-6 shows the contiguous block of `Words` included in the boundary specified by `Ex_bitmap.rect` and placed in the coordinate system that `Ex_bitmap.rect` establishes. (Each bit of a `Word` is represented by a grid square.)

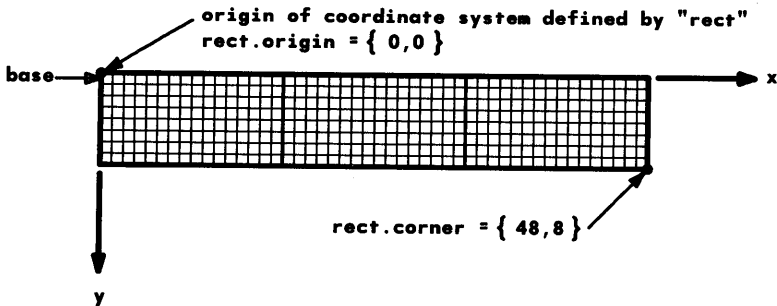


Figure 4-6: "rect" = {{0,0},{48,8}}

Four Basic Data Types

B. Consider the following **Bitmap** specification:

```
Bitmap Ex_bitmap={  
    &Word_5;  
    3,  
    {{16,16} , {64,24}},  
    (char *)0,  
};
```

The boundary specified by **rect** can be calculated as follows:

```
rect.corner.x - rect.origin.x = bits wide (64-16=48)  
rect.corner.y - rect.origin.y = bits high (24-16=8).
```

Ex_bitmap.rect still bounds the same image data. Only the position of the origin of the coordinate system established by **Ex_bitmap.rect** has changed. See Figure 4-7.

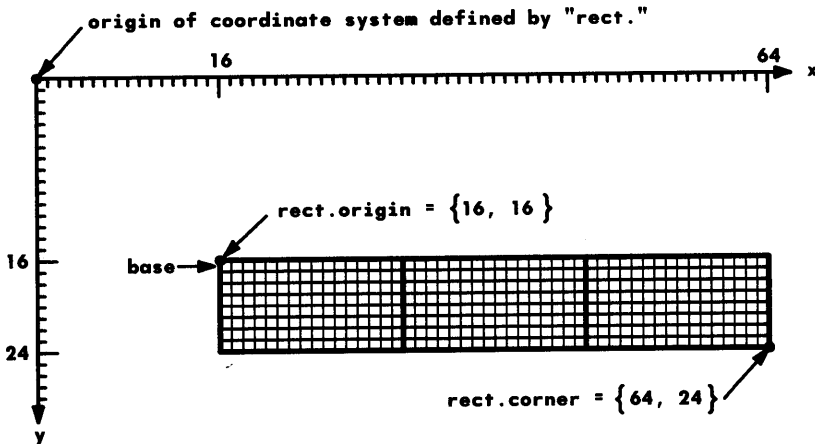


Figure 4-7: "rect" = {{16,16},{64,24}}

- C. To this point, care has been given to consider only those definitions of **rect** where **rect.origin.x** is divisible by 16 (the number of bits in a **Word**). When this is not the case, things get a little more interesting. Consider the following **Bitmap** specification:

```
Bitmap Ex_bitmap={
    &Word_5;
    3,
    {{10,10} , {42,16}},
    (char *)0,
};
```

The rectangle boundary specified by **Ex_bitmap.rect** is:

```
42 - 10 = 32 bits wide
16 - 10 = 6 bits high.
```

The left edge of **Ex_bitmap.rect** starts at bit 10 in the first **Word** of the image data. The bit offset of the left edge of the rectangle within the **Word** pointed to by **Ex_bitmap.base** is given by the following expression:

```
rect.origin.x % WORDSIZE
```

where "%" is the modulo operator. **WORDSIZE** is defined in the include file **dmd.h** and is equal to the number of bits in a **Word**. Bits of **Words** residing outside of the boundary defined by **Ex_bitmap.rect** are ignored by graphical operations. See Figure 4-8 for an illustration.

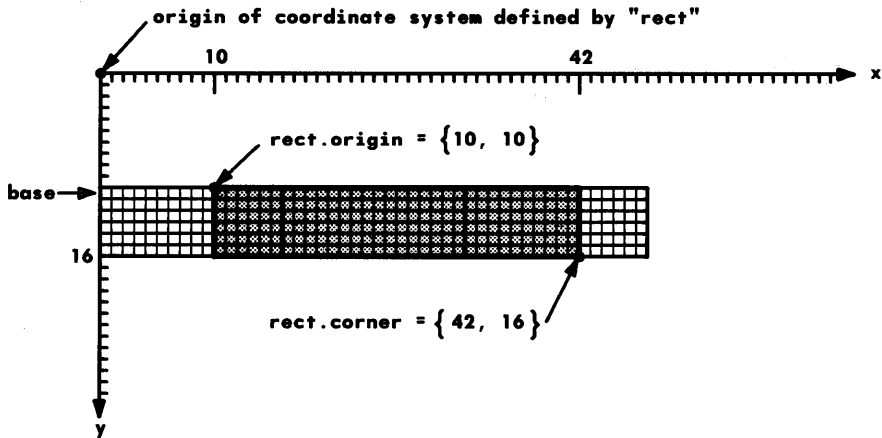


Figure 4-8: "rect" = {{10,10},{42,16}}

D. As a final example, consider the following **Bitmap** specification:

```

Bitmap Ex_bitmap={
    &Word_5;
    3,
    {{24, 5} , {44,10}},
    (char *)0,
};

```

The rectangle boundary specified by **Ex_bitmap.rect** is:

$$44 - 24 = 20 \text{ bit wide}$$

$$10 - 5 = 5 \text{ bits high.}$$

The offset of **Ex_bitmap.rect** within the **Word** referenced by **base** is:

$$24 \% 16 = 8 \text{ bits}$$

See Figure 4-9 for an illustration. Note that a full **Word** at the end of each scan line has been excluded by the **Ex_bitmap.rect** boundary. This can be a useful feature when defining **Bitmaps** that reside within larger **Bitmaps**; for example, the **Bitmaps** for windows residing within the larger **Bitmap** of the entire screen.

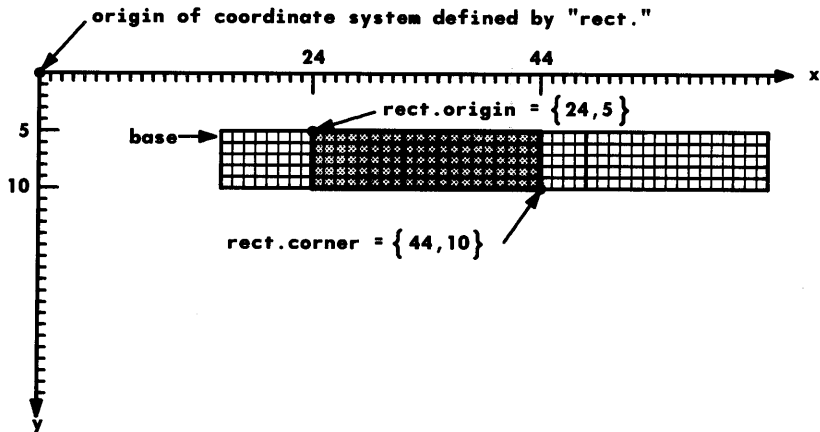


Figure 4-9: "rect" = {{24,5},{44,10}}

Include File "world.h"

For an example of an actual **Bitmap** definition, see the include file **world.h** used in the program **HELLO.c** in directory *\$DMD/examples/GettingStarted*. The array of **Words**, **world_bits**, defines the image data for the **Bitmap**. This image data was created using the **icon** application program with a 50 by 50 grid. See the Chapter "Icon Editing" of this document for more information. The **Bitmap** definition called **world** has **base** set to the first **Word** of the array **world_bits**.

width was calculated by determining the smallest integer number of **Words** needed to cover a single scan line of the image data. Since the image data is 50 bits wide, it takes four **Words** to cover a single scan line.

rect is defined with its origin at (0, 0) and corner point (50, 50). From this definition of **rect**, the image is bounded by a box that is 50 bits in the x-direction and 50 scan lines in the y-direction.

Four Basic Data Types

You should experiment with the **world** Bitmap definition and the example program **HELLO.c** to gain an understanding of how **width** and **rect** affect the image displayed on the screen. For example, try using the following Bitmap definition:

```
Bitmap world = {
    (Word *)world_bits,
    4,
    { { 12, 0 }, { 35, 25 } }
};
```

Operations, Comparisons and Conversions

Before getting into the drawing routines of the 630 MTG, it is appropriate to discuss some of the operations that manipulate the four abstract data types we have just defined. An understanding of these operations will enhance your ability to make use of the drawing routines.

Operations on "Points"

The operations on **Points** include arithmetic addition, subtraction, multiplication and division. Given two Points, "p1" and "p2", and an integer constant "a", these operations are defined as follows:

- Point Addition - $\text{add}(p1, p2)$ returns a Point p3 such that $p3.x = p1.x + p2.x$ and $p3.y = p1.y + p2.y$.
- Point Subtraction - $\text{sub}(p1, p2)$ returns a Point p3 such that $p3.x = p1.x - p2.x$ and $p3.y = p1.y - p2.y$.
- Point Multiplication - $\text{mul}(p1, a)$ returns a Point p3 such that $p3.x = a * p1.x$ and $p3.y = a * p1.y$.
- Point Division - $\text{div}(p1, a)$ returns a Point p3 such that $p3.x = p1.x / a$ and $p3.y = p1.y / a$.

The subtraction routine has a very interesting result when applied to the origin and corner of a Rectangle. For example, given a Rectangle "r", $\text{sub}(r.\text{corner}, r.\text{origin})$ returns a Point that would be the corner of the Rectangle "r" if the origin of "r" were translated to the Point (0, 0). This is often referred to as the **extent** of the Rectangle.

Operations, Comparisons and Conversions

Another application of these routines is a subroutine that returns the Point corresponding to the center of a Rectangle. This can be accomplished by dividing the extent of the Rectangle by two and adding the result to its origin. See the following example in Figure 4-10.

```
#include <dmd.h>

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

Point
GetRectCenter(r)
Rectangle r;
{
    Point HalfOfExtent;

    HalfOfExtent = div( sub(r.corner, r.origin), 2);
    return add(r.origin, HalfOfExtent);
}
```

Figure 4-10: Example Rectangle - "extent" Divided by Two

The Point arithmetic operations can be found on the manual page **PTARITH(3R)** in the *630 MTG Software Reference Manual*.

"Point" Comparison

Two Points can be compared for equality by using the routine **eqpt**. These Points are equal if the corresponding x- and y-coordinates are equal. **eqpt** returns a 1 if the Points are equal or a 0 if they are not. See the manual page **EQ(3R)** in the *630 MTG Software Reference Manual* for more details on **eqpt**.

“Rectangle” Operations

Arithmetic operations on Rectangles include addition and subtraction of a Point from a Rectangle. Given a Rectangle "r" and a Point "p", these two operations are defined as follows:

- Add Point to Rectangle - **raddp**(r, p) returns a Rectangle with origin equal to add(r.origin, p) and corner equal to add(r.corner, p).
- Subtract Point from Rectangle - **rsubp**(r, p) returns a Rectangle with origin equal to sub(r.origin, p) and corner equal to sub(r.corner, p).

An example application of these routines is shown in the routine **CenterRectOnPt** (Figure 4-11). This routine, when given a Rectangle "r" and a Point "p", will return a Rectangle centered on "p". The previous example, **GetRectCenter**, is used to determine the center of the Rectangle "r". The difference between the center point of the Rectangle and the Point "p" is then added to the Rectangle "r" to center it on "p".

```
#include <dmd.h>

Point GetRectCenter();
Rectangle raddp();

Rectangle
CenterRectOnPt(r, p)
Rectangle r;
Point p;
{
    Point RectCenter;

    RectCenter = GetRectCenter(r);
    return raddp(r, sub(p, RectCenter));
}
```

Figure 4-11: Example Rectangle - CenterRectOnPt

Operations, Comparisons and Conversions

Another very simple application of **raddp** and **rsubp** is a routine that translates the origin of a Rectangle "r" to the origin of some other coordinate system (Figure 4-12). The origin of that coordinate system need not be at (0, 0). This is accomplished by translating the rectangle to (0, 0) and then retranslating it to the Point "p".

```
#include

Rectangle rsubp();
Rectangle raddp();

Rectangle
MoveRectToOrigin(r, p)
Rectangle r;
Point p;
{
    return raddp(rsubp(r, r.origin), p);
}
```

Figure 4-12: Example Rectangle - "raddp" and "rsubp"

The **Rectangle** arithmetic operations can be found on the manual page **RECTARITH(3R)** in the *630 MTG Software Reference Manual*.

Two other operations on **Rectangles** include the routines **inset** and **rectclip**. The operation of these routines is documented fully on the respective manual pages, **INSET(3R)** and **RECTCLIP(3R)**, in the *630 MTG Software Reference Manual*.

"Rectangle" Comparison

Two Rectangles, "r1" and "r2", can be checked for equality by using the routine **eqrect**. Two Rectangles are equal if the corresponding origin and corner Points are equal. **eqrect** returns a 1 if the Rectangles are equal or a 0 if they are not. See the manual page **EQ(3R)** in the *630 MTG Software Reference Manual* for more details on **eqrect**.

Inclusion Operations

To ease the interaction of Points with Rectangles and Rectangles with Rectangles, two routines, **ptinrect** and **rectXrect**, have been provided. See the manual pages **PTINRECT(3R)** and **RECTXRECT(3R)** in the *630 MTG Software Reference Manual*. Given a Point "p" and a Rectangle "r", **ptinrect** checks to see if "p" is within the bounds of "r". This routine is very useful when an application is tracking the position of the mouse cursor and wants to determine if the mouse cursor position has moved into a particular Rectangle. More will be said about mouse tracking in the "Application Resources" Chapter.

Given two Rectangles, "r1" and "r2", **rectXrect** will determine whether the regions bounded by "r1" and "r2" overlap. If these regions do overlap, **rectXrect** returns a one; otherwise, it returns a zero.

Data Type Conversions

A number of data type conversion routines have been provided for the four basic graphical data types. These routines are documented in detail in the *630 MTG Software Reference Manual* and, therefore, only a short review will be given here.

- **addr** - given a Point "p" in a Bitmap "b", **addr** will return the address of the Word in "b" that contains "p". See the manual page **ADDR(3R)**.
- **fPt** - given two 16-bit integer coordinates, x and y, the function **fPt** will return a Point "p" such that p.x = x and p.y = y. See the manual page **FPT(3L)**.
- **Pt** - differs from **fPt** in that **Pt** is a macro and is meant to be used in an argument list for a function call. **Pt** will pass two 16-bit integers, x and y, as a Point to some function "f"; for example, add (Pt(3, 5), Pt(10, 25)). See the manual page **PT(3L)**.
- **canon** - given two Points, "p1" and "p2", the function **canon** will return a Rectangle "r" that has a positive "extent". In other words, **canon** will arrange the x- and y-coordinates of "p1" and "p2" so that the Rectangle returned has its defining Points, origin and corner, in standard form which is upper left and lower right. See the manual page **CANON(3R)**.

Operations, Comparisons and Conversions

- **fRpt** - similar to **canon** in that given two Points, "p1" and "p2", it will return a Rectangle "r". **fRpt** however will not make sure that the Rectangle has a positive "extent". **fRpt**(p1, p2) will return a Rectangle with "p1" as the origin and "p2" as the corner. See the manual page **FPT**(3L).
- **Rpt** - differs from **fRpt** in that it is a macro and is meant to be used in an argument list for a function call. **Rpt** will pass the two Points, "p1" and "p2", as a Rectangle to some function "f"; for example, `GetRectCenter(Rpt(p1, p2))`. See the manual page **PT**(3L).
- **fRect** - given four 16-bit integer coordinates, a, b, c, and d, **fRect** will return a Rectangle "r" with origin Point (a, b) and corner Point (c, d). See the manual page **FPT**(3L).
- **Rect** - differs from **fRect** in that it is a macro and is meant to be used in an argument list for a function call. **Rect** will pass four 16-bit integer coordinates as a Rectangle to some function "f"; for example, `GetRectCenter(Rect(a, b, c, d))`. See the manual page **PT**(3L).

Two Graphical Coordinate Systems

Two different coordinate systems exist in the 630 MTG graphical programming environment: screen coordinates and window coordinates. Both of these coordinate systems have the x-direction increasing from left to right and the y-direction increasing from top to bottom.

Screen coordinates refer to the actual pixels of the screen: the Point (0, 0) is the upper left corner, and (XMAX-1, YMAX-1) is the lower right corner of the screen. The constants XMAX and YMAX are defined in the include file **dmd.h** and are both set to 1024.

Window coordinates refer to a coordinate system that is confined to the rectangular portion of the screen used by an application program. The window coordinate system is scaled so that the Point (0, 0) corresponds to the upper left corner of the window, and the Point (XMAX-1, YMAX-1) corresponds to the lower right corner of the window. Since window coordinates are scaled, adjacent Points within the coordinate system will not necessarily refer to separate screen pixels.

Global Structures Describing the Two Coordinate Systems

There are several global data structures defined when the header file **dmd.h** is included to help the programmer deal with both screen and window coordinates.

- **physical** - This is a global Bitmap that describes the entire screen.
- **display** - This is a Bitmap that defines the display area available to an application program. It is the destination Bitmap most commonly used with the graphics routines. The **rect** field (defined by **display.rect**) of this Bitmap defines the Rectangle surrounding the window. This Rectangle also includes the window border. Note that **display.rect** is specified using the screen coordinate system.
- **Drect** - This global Rectangle defines the screen area inside the border of a window and is specified using screen coordinates.
- **Jrect** - The Rectangle **Jrect** is defined as {0, 0, XMAX, YMAX} and describes the screen area inside a window using window coordinates. **Jrect** does not include the border of the window.

Two Graphical Coordinate Systems

Note: The global structures **physical** and **display** are Bitmaps. **Drect** and **Jrect** are Rectangles. Also, **Drect** and **Jrect** describe the same Rectangle on the screen. The only difference is in the coordinate system used.

See Figures 4-13 and 4-14 for an illustration of these global data structures.

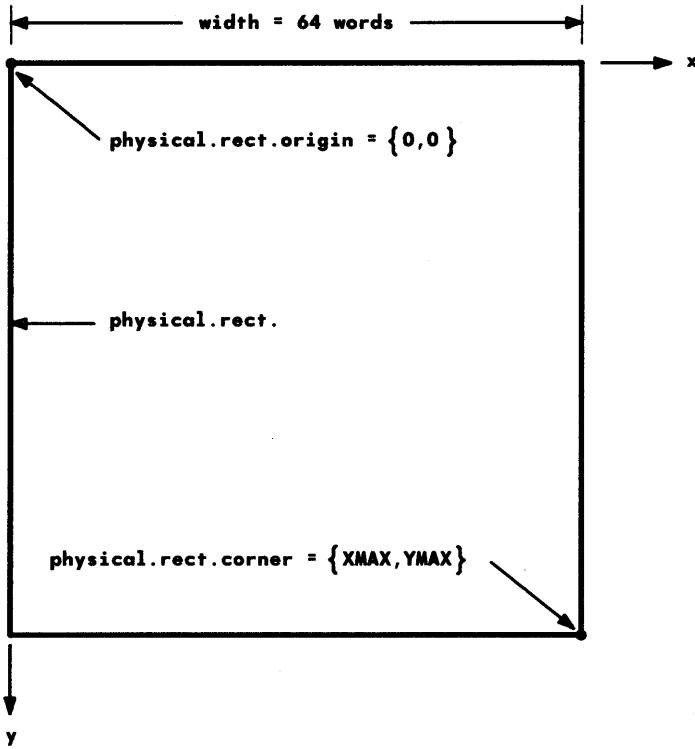


Figure 4-13: The Bitmap "physical"

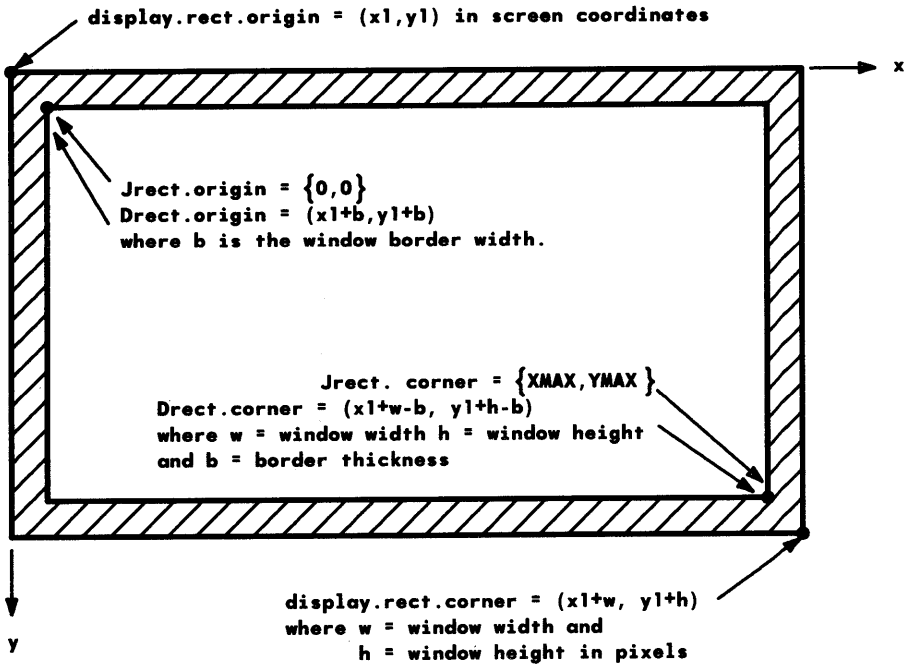


Figure 4-14: Global Structures Defining a Window

Current Point

In both the "screen" and "window" coordinate systems there is the concept of a current point within the application's window. This concept was mentioned briefly in the discussion of the `lprintf` routine in the "Getting Started" Chapter. There are actually two distinct current points maintained: one in screen coordinates and one in window coordinates. Modification of one has no effect on the other.

Current Point in the Screen Coordinate System

The current point for the screen coordinate system is available through the function **sPtCurrent** and can be moved using the routine **moveto**. See the manual page **MOVETO(3L)**. There are some subtleties about these two routines that are worth mentioning at this time.

moveto is a routine that allows you to move the current screen point to any position on the screen. However, if that current screen point falls outside your application's window (the rectangle defined by **Drect**), then any of the library routines that use the current screen point (such as **lprintf**) will reset the current screen point to **Drect.origin**. The current screen point can end up outside the boundaries of **Drect** as the result of a window **Reshape** or by specifically manipulating the current screen point using the routine **moveto**. If the window is moved, the current screen point moves accordingly.

Current Point in the Window Coordinate System

The current point for the window coordinate system is available through the global Point variable **PtCurrent**. See the manual page **GLOBALS(3R)**.

PtCurrent can be modified by using the routines **jmove** for relative movement or **moveto** for absolute movement. See the manual page **JMOVE(3L)**.

Coordinate System Transformations

It is possible that some applications will want to perform some manipulations using screen coordinates and other manipulations using window coordinates. Routines **transform** and **rtransform** have been provided to facilitate moving from the window coordinate system to the screen coordinate system.

- **transform** - given a Point "p" in window coordinates, **transform(p)** will return the corresponding screen coordinate Point.
- **rtransform** - given a Rectangle "r" in window coordinates, **rtransform(r)** will return the corresponding screen coordinates Rectangle.

These two routines are documented in the manual page **TRANSFORM(3R/3L)**.

Graphics Routines

Two sets of graphical routines have been provided in the 630 MTG Software Development Package: a set of routines for window coordinates and a set of routines for screen coordinates. The routines for drawing in window coordinates can be distinguished from those that draw in screen coordinates by the "j" prefix added to the names of the window coordinate routines. Objects drawn using the window coordinate routines discussed in this chapter will always be scaled to fit in the application's window (the Rectangle defined by **Jrect**). Objects drawn using the screen coordinate routines will be "clipped" to a particular Bitmap specified in the parameter list when the routine is called. "Clipping" means drawing only that part of the object that lies within the bounding rectangle of the destination Bitmap specified.

Function Codes

Most of the graphics routines take a "Code" argument to specify a logical function for use when drawing. Using C syntax, the values and meanings of the codes are:

VALUE	MEANING
F_OR	target = source
F_CLR	target & = ~source
F_XOR	target ^ = source
F_STORE	target = source

In other words, if some source image is to be copied to some congruent target image with Code F_OR, the result will be the bitwise OR of the source image with the contents of the target image.

Code F_XOR is the bitwise exclusive-OR of a source image with the contents of a target image. The exclusive-OR operation has some very useful properties. From the laws of Boolean algebra, you may recall the following properties:

1. Given the Boolean variables x, y, and z, it is true that $(x \text{ XOR } y) \text{ XOR } x = y$.
2. A second useful property is $(x \text{ XOR } y) \text{ AND } z = (x \text{ AND } z) \text{ XOR } (y \text{ AND } z)$.

3. And finally,
 $x \text{ XOR } x = 0.$

From the above properties, we can make the following statements:

- F_XOR is its own true inverse: two adjacent identical F_XOR operations cancel exactly, restoring the screen to its previous form.
- F_XOR is commutative: F_XOR operations may be executed in any order to produce the same final result. Combined with its inverse property, this means that an F_XOR operation may be canceled at any later time by another F_XOR operation.
- Because F_XOR is its own inverse, the same code can be used to draw or undraw a picture. This is a common action: the mouse cursor, for example, is updated by calling the same routine, using F_XOR mode internally, to undraw the old position and draw the new one (the order is irrelevant).

Of course, these properties can break down if F_XOR operations are mixed with other modes. However, it is not only possible, but common to do all graphics in F_XOR mode.

Drawing Routines

Graphical routines have been provided to draw and fill, when appropriate, the following objects: points (one pixel), lines, rectangles, circles, circular arcs, ellipses and elliptical arcs.

The following sections describe the different drawing routines available and the names of their respective manual pages. The *630 MTG Software Reference Manual* contains details and examples on the use of each of these routines.

Screen Coordinate Drawing Routines

The following screen coordinate routines operate on **Bitmaps** which are always specified in screen coordinates. For more information on each routine, refer to the appropriate manual page in the *630 MTG Software Reference Manual*.

Graphics Routines

- Rectangles

box - draws a rectangle.

See **BOX(3R)**.

rectf - performs functions on a rectangle.

See **RECTF(3R)**.

texture - fills a rectangle with a texture.

See the section on "Textures" in this chapter and the manual page **TEXTURE(3R)**.

- Circles

circle - draws a circle.

disc - draws a filled circle.

discture - draws a texture filled circle.

arc - draws an arc.

See **CIRCLE(3L)**.

- Ellipses

ellipse - draws an ellipse.

eldisc - draws a filled ellipse.

eldiscture - draws a textured filled ellipse.

elarc - draws an elliptical arc.

See **ELLIPSE(3L)**.

- Lines

segment - draws a line segment.

See **SEGMENT(3R)**.

- Points

point - draws a single pixel.

See **POINT(3R)**.

- Polygons

polyf - fills a polygon.

See **POLYGON(3L)**.

The example program **screen.c** in directory *\$DMD/examples/Graphics* provides a demonstration of several screen coordinate drawing routines. The source code for **screen.c** is given in Figure 4-24 at the end of this chapter.

Window Coordinate Routines

The window coordinate routines always operate within the global Bitmap "display".

- Circles

jcicle - draws a circle.

jdisc - draws a filled circle.

jarc - draws an arc.

See **JCIRCLE(3L)**.

- Ellipses

jellipse - draws an ellipse.

jeldisc - draws a filled ellipse.

jelarc - draws an elliptical arc.

See **JELLIPSE(3L)**.

- Lines

jsegment - draws a line segment.

See **JSEGMENT(3L)**

- Points

jpoint - draws a point.

See **JPOINT(3L)**.

■ Rectangles

jrectf - performs function in rectangle.

jtexture - draws a texture in rectangle.

See **JRECTF(3L)** and **JTEXTURE(3L)**.

The fundamental difference between the screen coordinate routines and the window coordinate routines is that the former allows the specification of a particular bitmap in which to draw the object and always uses the screen coordinate system. The window coordinate routines, on the other hand, always draw in the global Bitmap "display" and scale the drawing to the window coordinate system.

Textures

Texture16 is another data type defined in the include file **dmd.h**. The definition of **Texture16** is as follows:

```
typedef struct Texture16 {
    Word bits[16];
} Texture16;
```

A **Texture16** is an array of 16 Words or equivalently a 16 by 16 array of bits, which defines a dot pattern. For example, a **tweed** texture is declared as:

```
Texture16 tweed={
    0x4444, 0x7777, 0xEEEE, 0x2222,
    0x4444, 0x7777, 0xEEEE, 0x2222,
    0x4444, 0x7777, 0xEEEE, 0x2222,
    0x4444, 0x7777, 0xEEEE, 0x2222
};
```

A **Texture16** is much like a **Bitmap** that has a fixed **width** of one **Word**, a fixed bounding rectangle with dimensions 16 by 16, and a two-dimensional array of bits within the bounding rectangle that describes the image or texture. The first **Word** of the image is the first horizontal scan line of the **Texture16**, the second **Word** is the next scan line, and so on for 16 scan lines. The routines which use **Texture16s** fix the patterns to absolute screen coordinates so that, for example, if two overlapping screen rectangles are textured with the same **Texture16**, the dots in each rectangle will mesh properly to form a constant pattern.

The routine **texture** will fill a rectangle in a specified Bitmap with a texture. For example, the following program (Figure 4-15) will fill the global Bitmap **display** with the **tweed** texture defined above:

```
#include <dmd.h>

Texture16 tweed={
    0x4444, 0x7777, 0xEEEE, 0x2222,
    0x4444, 0x7777, 0xEEEE, 0x2222,
    0x4444, 0x7777, 0xEEEE, 0x2222,
    0x4444, 0x7777, 0xEEEE, 0x2222
};

main()
{
    request(KBD);
    texture(&display, Direct, &tweed, F_XOR);
    wait(KBD);
}
```

Figure 4-15: Example Program - Bitmap "display"

A few textures are used frequently by 630 MTG programs. These are stored in ROM and software libraries for fast access. Their names are:

T_background	(ROM)
T_darkgrey	(library)
T_lightgrey	(library)
T_grey	(ROM)
T_white	(ROM)
T_black	(ROM)
T_checks	(ROM)

A variation of the use of a **Texture16** is to define mouse cursors and graphics icons. The AT&T logo is the sample **Texture16** below.

```
Texture16 globe = {  
    0x07E0, 0x0000, 0x207C, 0x7FFE,  
    0x0000, 0x803F, 0xFFFF, 0x0000,  
    0xC07F, 0xFFFF, 0x0000, 0x7FFE,  
    0x7FFE, 0x0000, 0x1FF8, 0x07E0,  
};
```

The following cursors and icons are stored in ROM or software libraries.

```
C_crosshair    (library)  
C_sweep        (library)  
C_confirm      (library)  
C_clock        (ROM)  
C_move         (ROM)  
C_skull        (ROM)  
C_target       (ROM)  
C_cup          (ROM)  
C_deadmouse    (ROM)
```

For an example of using textures as cursors, see the "Application Resources" Chapter under the heading "The Mouse Resource."

For more detailed information on using **Texture16s**, see the "ICON Editing" Chapter in Part 2 of this document and the **ICON(1)** manual page in the *630 MTG Software Reference Manual*.

“bitblt”

bitblt is a routine that operates on **Bitmaps**. See the manual page **BITBLT(3R)** for details. Basically, it will take a rectangular image from one **Bitmap** in memory and copy it into another **Bitmap** in memory. The **bitblt** routine is declared as follows:

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

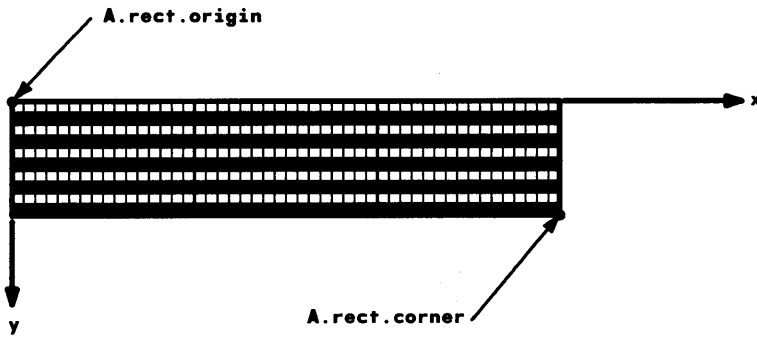
bitblt copies valid image data within Rectangle "r" from source Bitmap "sb" where "r" is specified within the coordinate system defined by "sb->rect", to a congruent Rectangle with origin "p" in the destination Bitmap "db" where "p" is specified within the coordinate system defined by "db->rect". The nature of the copy is specified by the function Code "f". The source and destination Bitmaps may be the same, and the source and destination Rectangles may overlap.

Note: There is no **jbitblt** -- **bitblt** operates directly on Bitmaps which are always specified in the screen coordinate system.

The illustrations in the following section will clarify this operation.

“bitblt” Illustrations

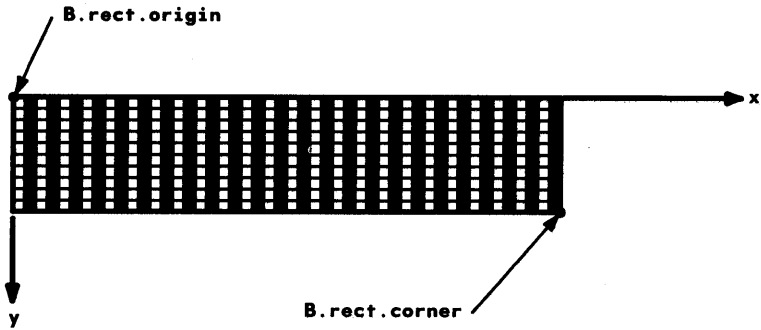
Consider the two Bitmaps shown in Figures 4-16 and 4-17. The image data in Bitmap "A" is a series of horizontal lines. The image data in Bitmap "B" is a series of vertical lines. Both Bitmaps are shown within their respective coordinate systems defined by the Rectangles "A.rect" and "B.rect".



```
Word  A_image_data[ ] = {  
    0x0000, 0x0000, 0x0000,  
    0xFFFF, 0xFFFF, 0xFFFF,  
    0x0000, 0x0000, 0x0000,  
    0xFFFF, 0xFFFF, 0xFFFF,  
    0x0000, 0x0000, 0x0000,  
    0xFFFF, 0xFFFF, 0xFFFF,  
    0x0000, 0x0000, 0x0000,  
    0xFFFF, 0xFFFF, 0xFFFF,  
    0x0000, 0x0000, 0x0000,  
    0xFFFF, 0xFFFF, 0xFFFF,  
};
```

```
Bitmap A = {  
    A_image_data;  
    3,  
    {{ 0,0 }, { 48,10 }},  
    (char*)0,  
};
```

Figure 4-16: Bitmap A



```

Word  B_image_data[ ] = {
    0x5555, 0x5555, 0x5555,
    0x5555, 0x5555, 0x5555,
    0x5555, 0x5555, 0x5555,
    0x5555, 0x5555, 0x5555,
    0x5555, 0x5555, 0x5555,
    0x5555, 0x5555, 0x5555,
    0x5555, 0x5555, 0x5555,
    0x5555, 0x5555, 0x5555,
    0x5555, 0x5555, 0x5555,
    0x5555, 0x5555, 0x5555,

```

```
};
```

```

Bitmap B = {
    B_image_data,
    3,
    {{0,0}, {48,10}},
    (char*)0,

```

```
};
```

Figure 4-17: Bitmap B

Graphics Routines

Executing the following `bitblt` command will generate the results shown in Figure 4-18.

```
bitblt(&A, Rect(0,0,16,10), &B, B.rect.origin, F_STORE);
```

Note: The source Bitmap is not altered by a `bitblt`.

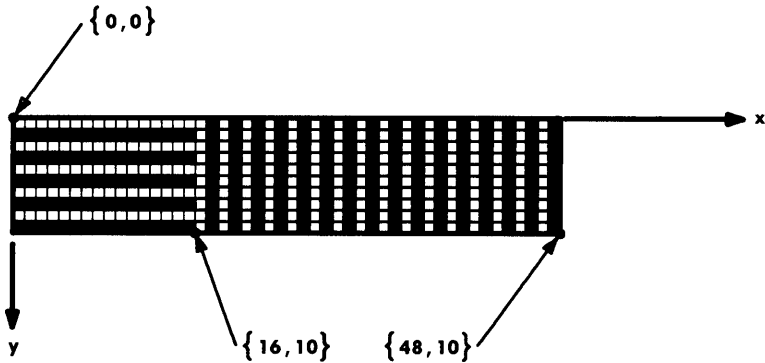


Figure 4-18: "bitblt" - A Modifying the Image Data of B

Figure 4-19 shows the results of the following `bitblt` command:

```
bitblt(&B, Rect(16, 5, 24, 10), &A, Pt(8, 0), F_STORE);
```

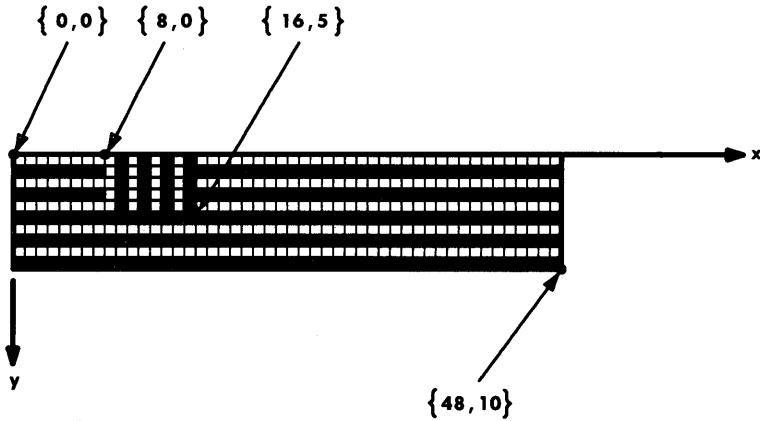
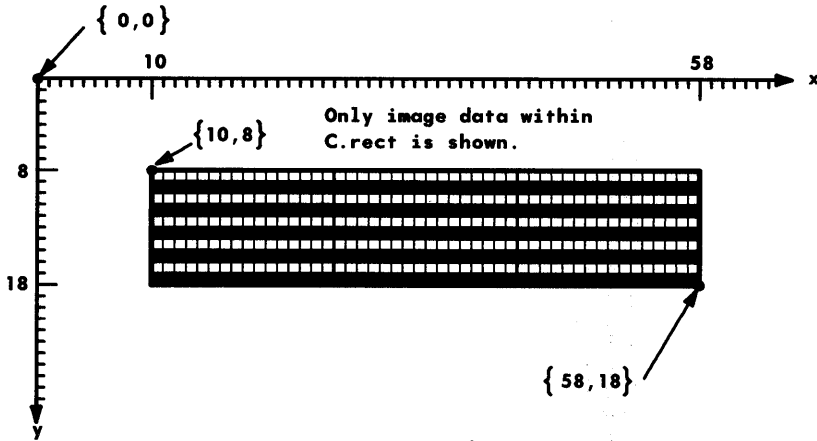


Figure 4-19: "bitblt" - B Modifying the Image Data of A

Figures 4-20 and 4-21 show the definition of Bitmaps "A" and "B" modified slightly and renamed as "C" and "D", respectively. Note that the illustration of the Bitmaps "A" and "B" show only the valid image data within the rectangles "C.rect" and "D.rect".



```

Word C_image_data[ ] = {
    0x0000, 0x0000, 0x0000, 0x0000,
    0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
    0x0000, 0x0000, 0x0000, 0x0000,
    0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
    0x0000, 0x0000, 0x0000, 0x0000,
    0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
    0x0000, 0x0000, 0x0000, 0x0000,
    0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
    0x0000, 0x0000, 0x0000, 0x0000,
    0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
};

Bitmap C = {
    C_image_data,
    4,
    {{10,8}, {58,18}},
    (char*)0,
};
    
```

Figure 4-20: Bitmap A Modified as C

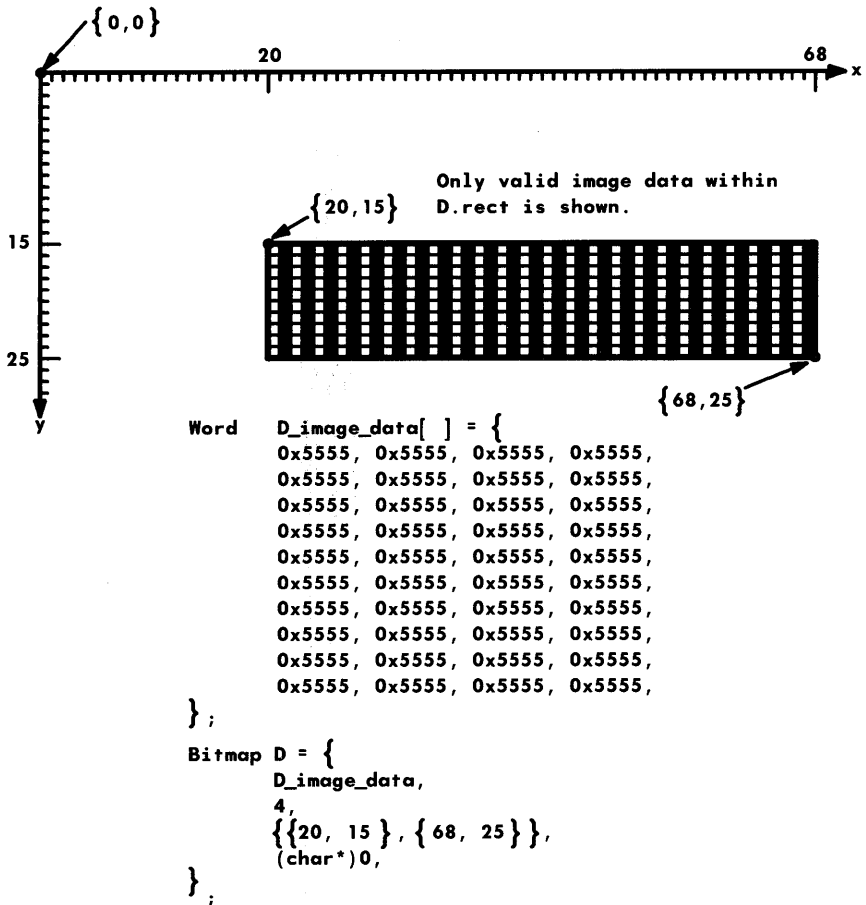


Figure 4-21: Bitmap B Modified as D

Graphics Routines

Executing the following **bitblt** command will generate the results shown in Figure 4-22:

```
bitblt(&C,Rect(8, 5, 28, 24), &D, Pt(36, 8), F_STORE);
```

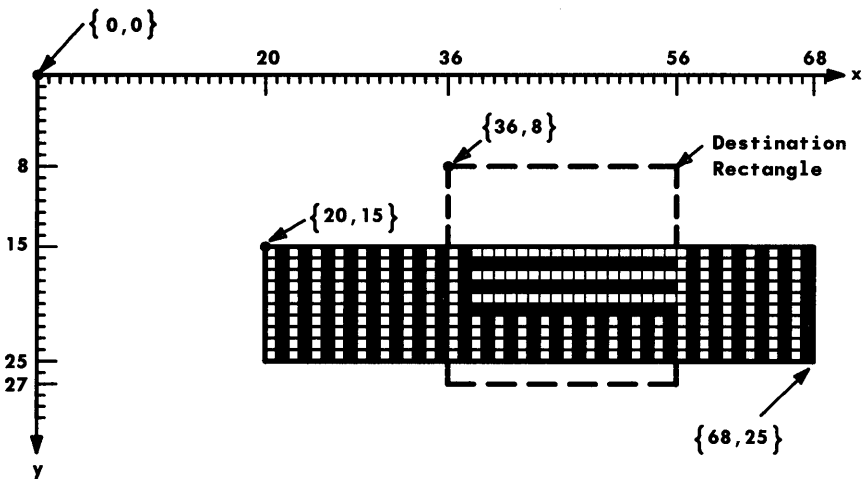


Figure 4-22: "bitblt" - C Modifying Image Data of D

Using "bitblt" - "twist.c"

A common use of **bitblt** is to copy a prepared picture from off-screen onto the 630 MTG screen. The example program **twist.c** in directory *\$DMD/examples/Graphics* demonstrates how **bitblt** can be used to create the impression of a rotating world. The source code for **twist.c** is given in Figure 4-23.

```
#include <dmd.h>
#include "rotate.h"

/* Library Routines and associated manual page. */
void bitblt();          /* BITBLT(3R)          */
void lprintf();        /* PRINTF(3L)          */
void sleep();          /* SLEEP(3R)           */
Point sPtCurrent();    /* MOVETO(3L)          */
int wait();            /* RESOURCES(3R)       */

main()
{
    int i;
    Point savept;

    /*
    ** Use "lprintf" to move the "current screen point"
    ** and set a position at which to display the
    ** image of the world.
    */
    lprintf("\n          ");
    savept = sPtCurrent();

    lprintf("\n Hello          !");

    for (i=0; i < 1 ; i++) {
        /*
        ** after the 18th Bitmap, go back to Bitmap zero.
        */
        if(i==18)
            i=0;
    }
}
```

Figure 4-23: Source Code for "twist.c" (Sheet 1 of 2)

Graphics Routines

```
    /*
    ** Draw the current Bitmap of the world.
    */
    bitblt(world[i], world[i]->rect, &display, savept, F_XOR);

    /*
    ** Release the CPU for 10 ticks of the
    ** 60 Hz system clock.
    */
    sleep(10);

    /*
    ** Erase the current Bitmap of the world in
    ** preparation for drawing the next.
    */
    bitblt(world[i], world[i]->rect, &display, savept, F_XOR);
}
}
```

Figure 4-23: Source Code for "twist.c" (Sheet 2 of 2)

Example Program - "screen.c"

Figure 4-24 gives the source code for `screen.c`, which demonstrates several of the screen coordinate drawing routines.

```
#include <dmd.h>

/* Library Routines and associated manual page. */
Point add();          /* PTARITH(3R)          */
void box();          /* BOX(3R)            */
void circle();       /* CIRCLE(3R)         */
void disc();         /* CIRCLE(3R)         */
void discture();    /* CIRCLE(3R)         */
Point div();         /* PTARITH(3R)        */
Point fPt();         /* FPT(3L)            */
Rectangle inset();  /* INSET(3R)          */
int min();           /* INTEGER(3R)        */
void point();        /* POINT(3R)          */
void polyf();        /* POLYGON(3L)        */
void segment();     /* SEGMENT(3R)        */
Point sub();         /* PTARITH(3R)        */
int wait();          /* RESOURCES(3R)      */

/* Local functions declared in this file */
Point GetRectCenter();

/*
** Define the number of vertices in polygons.
*/
#define POLYPOINTS      4

main()
{
    int Radius;
    Point WindowCenter, DrectExtent;
    Rectangle MyRect;

    /*
    ** Declare three polygons. A polygon is defined by an array
    ** of Points. Each polygon in this example has four vertices.
    */
    Point poly1[POLYPOINTS];
    Point poly2[POLYPOINTS];
    Point poly3[POLYPOINTS];
}
```

Figure 4-24: Example Program "screen.c" (Sheet 1 of 3)

Example Program - "screen.c"

```
/*
** Get the center of the Rectangle Drect and also
** determine its extent.
*/
WindowCenter = GetRectCenter(Drect);
DrectExtent = sub(Drect.corner, Drect.origin);

/*
** Calculate the radius of a circle.
*/
Radius = min(DrectExtent.x, DrectExtent.y) / 4;

/*
** Draw a circle, a texture filled circle,
** a filled circle, and a Point.
*/
circle (&display, WindowCenter, Radius, F_XOR);
discture (&display, WindowCenter, 2*Radius/3, &T_checks, F_XOR);
disc (&display, WindowCenter, Radius/3, F_STORE);
point(&display, WindowCenter, F_XOR);

/*
** Draw a rectangle.
*/
MyRect = inset(Drect, Radius);
box(&display, MyRect, F_XOR);

/*
** Initialize the points in polygon one.
*/
poly1[0] = Drect.origin;
poly1[1] = MyRect.origin;
poly1[2] = fPt(MyRect.origin.x, MyRect.corner.y);
poly1[3] = fPt(Drect.origin.x, Drect.corner.y);

/*
** Initialize the points in polygon two.
*/
poly2[0] = fPt(Drect.origin.x, Drect.corner.y);
poly2[1] = fPt(MyRect.origin.x, MyRect.corner.y);
poly2[2] = MyRect.corner;
poly2[3] = Drect.corner;
```

Figure 4-24: Example Program "screen.c" (Sheet 2 of 3)

```
/*
** Initialize the points in polygon three.
*/
poly3[0] = fPt(Direct.corner.x, Direct.origin.y);
poly3[1] = fPt(MyRect.corner.x, MyRect.origin.y);
poly3[2] = MyRect.corner;
poly3[3] = Direct.corner;

/*
** Draw the three texture filled polygons.
*/
polyf(&display, poly1, POLYPOINTS, &T_black, F_XOR);
polyf(&display, poly2, POLYPOINTS, &T_grey, F_XOR);
polyf(&display, poly3, POLYPOINTS, &T_black, F_XOR);

/*
** Draw four line segments.
*/
segment(&display, poly1[0], poly1[2], F_XOR);
segment(&display, poly1[1], poly1[3], F_XOR);
segment(&display, poly3[0], poly3[2], F_XOR);
segment(&display, poly3[1], poly3[3], F_XOR);

/*
** Share the CPU with other applications
** and 630 MTG system processes.
*/
for (;;) wait(CPU);
}

/*
** Function to find the center of a Rectangle.
*/
Point
GetRectCenter(r)
Rectangle r;
{
    Point HalfOfExtent;

    HalfOfExtent = div( sub(r.corner, r.origin), 2);
    return add(r.origin, HalfOfExtent);
}
```

Figure 4-24: Example Program "screen.c" (Sheet 3 of 3)

Chapter 5: Application Resources

Application Resource Management	5-1
Requesting a Resource - "request"	5-1
Servicing a Resource - "own"	5-2
Waiting on a Resource - "wait"	5-3
The Mouse Resource	5-4
The Global Structure "mouse"	5-4
Button Interface Macros	5-5
Mouse Tracking and "Button" Interface Macros	5-5
Button Interface Function - "bttns"	5-6
Mouse Cursor Control	5-6
Turning the Mouse Cursor Off and On - "cursinhibit" and "curallow"	5-6
Using Different Mouse Cursors - "cursswitch"	5-7
Setting the Mouse Position - "cursset"	5-7
Drawing with the Mouse	5-7
Graphic Routine with a Built-in Mouse Interface - "newrect"	5-7
The Keyboard Resource - "kbdchar" and "ringbell"	5-8
The Printer Resource	5-9
Sending Data to the Printer - "psendchar" and "psendchars"	5-9
Using the Printer and Keyboard as a Typewriter	5-9
Host Communications	5-10
Receiving and Sending Data - "rcvchar" and "sendchar"	5-10
A Simple Terminal Emulator - "vsterm1.c"	5-11
Releasing the Host Connection - "local"	5-11
Regaining the Host Connection - "attach"	5-11
Making Local Copies of Applications - "peel"	5-12

Chapter 5: Application Resources

System Services	5-13
CPU Resource	5-13
DELETE Resource	5-13
RESHAPED Resource	5-14
ALARM Resource - "alarm"	5-14
MSG Resource	5-15
 Example Programs	 5-17

Application Resource Management

This chapter describes how to manage and use the 630 MTG application resources. Application resources are I/O devices (such as the keyboard and the mouse) and system services. System services are simply flags set by the 630 MTG operating system indicating the occurrence of a particular event, such as the application window being reshaped or deleted. A complete list of 630 MTG application resources and management routines can be found on the manual page **RESOURCES(3R)** in the *630 MTG Software Reference Manual*.

Requesting a Resource - "request"

request is a routine that announces an application's intent to use one or more resources. Since a resource must be requested before it is used, **request** is most commonly called early in the application. The declaration of **request** is shown below:

```
int request(r)
int r;
```

"r" is a bit vector that represents which resources are being requested and is composed of the inclusive OR of a set of predefined masks each corresponding to a specific resource. The predefined masks are listed below with the resource they represent.

<u>MASK</u>	<u>Represented Resource</u>
MOUSE	mouse buttons and cursor position
KBD	characters received from the 630 MTG keyboard
PSEND	send characters to the printer
SEND	send characters from the 630 MTG to the host
RCV	characters received by the 630 MTG from host process
CPU	630 MTG cpu
DELETE	application is being deleted
RESHAPED	window has been reshaped or moved
ALARM	alarm has "fired"
MSG	state of message queue has changed

Application Resource Management

For example, an application that wants to use the mouse, keyboard, and printer would call **request** as shown below:

```
int resources;  
resources = request(KBD|MOUSE|PSEND);
```

request returns a bit vector indicating which resources have been granted. You should not assume that resources requested are automatically granted. For example, if one application program requests and is granted the printer resource, requests for the printer by other applications will fail since only one application at a time can be granted the printer. To determine if the printer resource has been granted, you can test the returned value from **request** as shown below:

```
int resources;  
resources = request(KBD|MOUSE|PSEND);  
  
if (resources&PSEND)  
    {printer granted};  
else  
    {printer in use by another application};
```

Each call to **request** overrides all previously requested resources. This means that all resources not specified in the latest **request** are not available to the application.

Servicing a Resource - "own"

own is a routine that returns a bit vector indicating which of the resources you requested are ready to be serviced. The declaration of **own** is shown below:

```
int own()
```

For example, if you requested the keyboard resource and want to determine if any characters have been typed, you could use the code fragment shown below:

```
if (own()&KBD)  
    {service keyboard}
```

Waiting on a Resource - "wait"

wait is a routine that allows your application to release the CPU, allowing other applications to run, until one of your requested resources is ready for service. **wait** returns a bit vector indicating which resources are ready for service. The declaration of **wait** is shown below:

```
int wait(r)  
int r;
```

The bit vector "r" specifies which resources you want to wait on. For example, if your application program has requested the keyboard and Mouse resources, the inner loop of your program may look something like the following:

```
int NeedsService;  
for(;;) {  
    NeedsService = wait(KBD|MOUSE);  
    if (NeedsService & KBD)  
        ProcessKbdChar(kbdchar());  
    if(NeedsService & MOUSE)  
        GetMouseStatus();  
}
```

The call to **wait** will suspend the above application until either the Mouse or Keyboard resource is ready for service.

The Mouse Resource

In many 630 MTG applications, the mouse is the primary means of selecting and directing application operations. This section describes how an application can interact with the mouse.

The Mouse is a 630 MTG application resource and, as such, must be requested before it is used. The line of code below shows how to request both the Mouse and the Keyboard resource:

```
request(MOUSEKBD);
```

A request for the Mouse resource will never be denied. This implies that the mouse will need to be shared by all the applications that have requested its use. The following code can be used to determine if your application is the current owner of the mouse:

```
if (own(&MOUSE)
    {got the mouse})
```

The condition `own(&MOUSE)` will be true only if your application is running in the current window and the mouse cursor is within an unobscured portion of that window.

The Global Structure "mouse"

Mouse button status and position are maintained for each application in a global data structure called `mouse`. The declaration of this structure is shown below:

```
struct Mouse {
    Point    xy, jxy;
    short    buttons;
};
```

The `Point xy` holds the current position of the mouse in the screen coordinate system, and `jxy` holds its position in the window coordinate system (see the "Graphics Environment" Chapter for more details on the screen and window coordinate systems). Current mouse button status is stored in the three least significant bits of `buttons`. Bits zero, one, and two correspond to mouse buttons three, two, and one (right, middle, and left), respectively. Button status is most easily interpreted through the button interface macros and functions discussed in the following section.

Button Interface Macros

The macros **button1**, **button2**, **button3**, **button12**, **button13**, **button23**, and **button123** can be used to determine mouse button status when the statement **own0&MOUSE** is true. This statement being true means your application is running in the current window and the mouse cursor is in an unobscured portion of that window. The macros will return a nonzero number if any of the buttons specified by the numbers in the macro name are depressed. Otherwise, the value returned will be zero. For example, if mouse button 3 is depressed and the conditions specified above are true, then the macros **button3**, **button23**, and **button123** will return non-zero values.

The macros **btn1**, **btn2**, **btn3**, **btn12**, **btn13**, **btn23**, and **btn123** can be used to determine mouse button status whenever your application is running in the current window. The mouse cursor does not need to be in your application's window. The macros will return a non-zero number if any of the buttons specified by the numbers in the macro name are depressed. Otherwise, the value returned will be zero. For example, **btn2()** returns a non-zero value whenever mouse button 2 is depressed and your application is running in the current window.

Note: It is strongly recommended that applications always use the **button** interface macros (rather than **btn**) since **button** macros report button status only when the mouse cursor is in the application's window. Whenever the mouse cursor is outside of your application's window, the 630 MTG system control process (which also uses the mouse) will be competing with your application for ownership of the mouse resource.

Mouse Tracking and "Button" Interface Macros

The program **mouse.c**, in the directory *\$DMD/examples/Resources*, demonstrates how the mouse button interface macros work. Figure 5-1 at the end of this chapter gives the source code for **mouse.c**. Compile and download this program using the following commands:

```
dmdcc -o mouse mouse.c
```

```
dmdld mouse
```

Download the compiled version of **mouse.c** (**mouse**) into a window that is at least 24 rows by 80 columns (with the Large Font). The program prints the names of the button interface macros in a table and, as mouse buttons are

depressed, highlights the macro names that return a non-zero value based on current button status and mouse cursor position. The program also displays the mouse cursor position in both the screen and window coordinate systems.

Note: This program does not release the CPU to allow other processes to run. This is to demonstrate the **bttn** macros without competition from the system control process.

Clicking button 1 in the box in the lower right corner of the application's window will cause **mouse.c** to exit.

Button Interface Function - "bttns"

The routine **bttns** is used to detect a change in mouse button status. When called with an argument of 0, **bttns** will loop without releasing the CPU until all mouse buttons are released. If called with an argument of 1, **bttns** will loop until at least one mouse button is depressed. Any other argument will cause **bttns** to return immediately.

Further details on the mouse button macros and functions can be found in the manual page **BUTTONS(3R/3L)** of the *630 MTG Software Reference Manual*.

Mouse Cursor Control

Using the mouse cursor can greatly enhance the graphics interface of an application program. It allows the user of the application to control program execution by pointing to and drawing objects on the screen. It also allows the application program to indicate the need for certain user actions by switching the mouse cursor **icon**.

Turning the Mouse Cursor Off and On - "cursinhibit" and "cursallow"

The routine **cursinhibit** turns off the mouse cursor when it moves into your application's window. This is useful when you want to track the mouse with some other graphical object other than one of the system mouse cursors. The example program **TrackMouse.c** tracks mouse movement with the Bitmap of the world used in the example program **hello.c** in the "Getting Started" Chapter. A printout of **TrackMouse.c** is shown in Figure 5-2 at the end of this chapter.

The routine **curallow** will enable mouse cursor tracking after a call to **cursinhibit**. There must be one call to **curallow** for every call to **cursinhibit** to enable normal mouse cursor tracking. See the manual page **CURSOR(3R)** for more details.

Using Different Mouse Cursors - “cursswitch”

Mouse cursors are Texture16 data types, normally referred to as icons. The normal mouse cursor is the arrow icon. The routine **cursswitch** can be used to change the mouse cursor to any Texture16. The example program on the manual page **CURSOR(3R)** demonstrates how to switch mouse cursors. The manual page also gives a list of available mouse cursors stored in the 630 ROMs. For details on how to create your own mouse cursors, see the Chapter "Icon Editing" of this document.

Setting the Mouse Position - “cursset”

The routine **cursset** allows your application to set the mouse position to any point on the screen. See the **CURSOR(3R)** manual page for more details.

Drawing with the Mouse

The demo program **star.c** is a good example of how the mouse can be used to create interactive graphics. **star.c** allows the user to select an initial point in the window by depressing mouse button 1 and then, while keeping button 1 depressed, will draw line segments between the current mouse position and the initial point selected. To make the pattern more interesting, the lines are reflected into the four different quadrants of the window. Clicking mouse button 2 will erase the contents of the window and allow the user to start a new drawing. A printout of **star.c** is shown in Figure 5-3 at the end of this chapter.

Graphic Routine with a Built-in Mouse Interface - “newrect”

The graphics routine **newrect** allows a user to use the mouse to interactively sweep out a rectangle on the screen. The return value is the swept rectangle. See the manual page **NEWRECT(3R)** in the *630 MTG Software Reference Manual* for further details.

The Keyboard Resource - “kbdchar” and “ringbell”

The keyboard is a resource that can be requested by multiple applications, but only the application running in the current window can receive keyboard input. An application can request the keyboard by executing:

```
request(KBD);
```

Once the keyboard is requested, characters typed will be placed on the application’s keyboard queue and can be retrieved one character at a time by calling **kbdchar**. **kbdchar** will return the next character from the keyboard queue or -1 when the queue is empty. The code fragment shown below will loop until the character ‘q’ is typed on the keyboard:

```
do
    wait(KBD);
while ( kbdchar() != 'q' );
```

Note: If an application does not request the keyboard, all characters typed will be sent to the host computer.

An application can ring the keyboard bell by calling the routine **ringbell**.

For further details, see the manual pages **KBDCHAR(3R)** and **RINGBELL(3R)** in the *630 MTG Software Reference Manual*. Also see the Chapter "Redefining the Keyboard" for information on the more advanced features of the keyboard.

The Printer Resource

The printer is a 630 MTG resource that can be owned by only one application at a time. When an application issues a request for the printer, it should test the bit vector returned by **request** to see if the request succeeded.

Sending Data to the Printer - “psendchar” and “psendchars”

The routines **psendchar** and **psendchars** can be used to send individual characters and strings of characters to the printer, respectively. **psendchar** will return 1 if successful and 0 if the printer output queue is full. **psendchars**, on the other hand, will not return to the calling application until it has succeeded in sending the entire character string. **psendchars** will relinquish the CPU if it has to wait for room on the printer output queue. Therefore, an application should use **psendchar** if it wishes to maintain ownership of the CPU while sending information to the printer.

The routines **xpsendchar** and **xpsendchars** operate in the same way as **psendchar** and **psendchars** except that **Terminal Setup** values for expanding tabs and filtering escape sequences are used when sending data to the printer. For more information on the **Terminal Setup** options, "Expand Tabs" and "Filter Escapes," see the Chapter "Terminal Setup" in the *630 MTG Terminal User's Guide*.

Using the Printer and Keyboard as a Typewriter

The example program **type.c** will enable the keyboard and a printer attached to the 630 MTG to act together as a typewriter. A printout of the source code for **type.c** is shown in Figure 5-4 at the end of this chapter.

For more information on the printer resource, see the manual pages **RESOURCES(3R)**, **PSENDCHAR(3R)**, and **PRINTQ(3R)**.

Host Communications

Applications running in the 630 MTG may have a connection with the host computer. This connection is available for sending and receiving host data unless the connection is specifically released (see the section below "Releasing the Host Connection"). To send and receive host data, the application must first request the send and receive resources as follows:

```
request (SENDRCV) ;
```

If the application wishes to send or receive exclusively, it can just request the associated resource. For example,

```
request (RCV) ;
```

will allow the application to receive data from the host.

Receiving and Sending Data - "rcvchar" and "sendchar"

Data received from the host is placed in the application's receive queue and read one character at a time by calling the routine **rcvchar** which returns the next character from the receive queue or -1 if the queue is empty. When the receive queue is empty, the application can suspend itself by executing

```
wait (RCV) ;
```

which is an instruction to wait until a character has been received from the host.

sendchar and **sendchars** send a single character or character string, respectively, to the host by placing data on the 630 MTG's output queue. If the output queue becomes full, **sendchar** and **sendchars** will wait until there is room on the output queue before returning to the calling application. This process is transparent to the calling application, but programmers should be aware that these two routines may not return immediately. See the manual pages **RCVCHAR(3R)** and **SENDCHAR(3R)** for more details.

A Simple Terminal Emulator - “vstern1.c”

vstern1.c in *\$DMD/examples/Resources*, is a simple terminal emulator that handles receiving characters from the host and displaying them on the screen. Figure 5-5 gives the source code for **vstern1.c**. It also does some very simple processing for the backspace character and rings the keyboard bell when it receives the ASCII BEL character. Note that **vstern1.c** does not request the keyboard resource. This means that characters typed when **vstern1.c** is the current window will be sent to the host.

Releasing the Host Connection - “local”

When an application is downloaded into the 630 MTG, it retains its connection for communications with the host. Many applications, however, do not need to communicate with the host and are therefore tying up a host connection that could be used by another application. For this reason, the routine **local** is provided for an application to release its host connection and run "locally" in the 630 MTG without host interaction. When an application calls **local**, the host connection that was used for the download is released for use by other applications in the terminal. The processes running on the host computer are terminated just as if the application's window had been deleted. In order to differentiate applications that are running locally from those that have a host connection, the border of the local application is changed to a textured pattern. For more details, see the manual page **LOCAL(3R)** in the *630 MTG Software Reference Manual*.

Regaining the Host Connection - “attach”

An application that had previously released its host connection can reestablish host communications by calling the routine **attach**. This routine will allocate a host connection for the application, if one is available, and change the application's border back to a solid outline. If there are no channels available for the requested host, the application already has a host connected, or the host argument is invalid, then **attach** will fail. This is indicated by a return value of zero. See the manual page **ATTACH(3R)** in the *630 MTG Software Reference Manual* for further details.

Making Local Copies of Applications - “peel”

An interesting combination of the features of **local** and **attach** has been provided in the routine **peel**. This routine will create a local copy of your application in a new window and leave the original copy attached to the host. Also, this routine can remove the original copy and start up the 630 MTG's default terminal emulator in the original window. See the manual page **PEEL(3R)** in the *630 MTG Software Reference Manual* for more details.

System Services

The remaining application resources behave somewhat differently from the I/O devices described so far. System services can be requested by an application and are simply flags set by the 630 MTG operating system, indicating occurrence of a particular event. These services are described in the following paragraphs.

CPU Resource

The **wait** function is typically used to allow other applications to run while waiting for some system resource to become available. The **CPU** resource is used with the **wait** to unconditionally give up the processor. In this case, **wait** will return after all other applications have had the opportunity to run. A typical application should periodically perform the **wait(CPU)** instruction if it executes for a long period of time; otherwise, all other applications are blocked from running. The **CPU** resource is always implicitly requested.

DELETE Resource

Requesting the **DELETE** resource prohibits a user from deleting your application with the main button 3 menu item **Delete**. Rather, a flag is set that causes a call to **own** to indicate that the **DELETE** resource is ready for service. This is intended for use by applications which wish to perform some type of cleanup before being deleted. If the **DELETE** resource does become ready for service, it is the responsibility of the application to perform the cleanup and delete itself. For an example, see the manual page **RESOURCES(3R)** in the *630 MTG Software Reference Manual*.

RESHAPED Resource

A call to the **wait** function with the **RESHAPED** resource will suspend the application until the application's window has been either moved or reshaped. A typical situation where this might be used is with an application which requires a certain minimum sized window. Upon determining that its current window size is too small, an application may wait until the user reshapes the window to an appropriate size:

```
while ((Direct.corner.x - Direct.origin.x) < MIN_WIDTH) {
    string(&mediumfont, "please reshape", &display,
        add(Direct.origin, Pt(10, 10)), F_XOR);
    wait(RESHAPED);
    P->state &= ~RESHAPED;
}
```

Note in the example that it is the responsibility of the application to clear the reshaped bit in the process state variable. This bit is used to determine if the **RESHAPED** resource is ready for service. Subsequent **wait(RESHAPED)** calls will return immediately if the application does not clear the **RESHAPED** bit. The **RESHAPED** resource is always implicitly requested. See the manual pages **RESOURCES(3R)** in the *630 MTG Software Reference Manual* for more details and examples of the **RESHAPED** resource, and see the manual page **STATE(3R)** for details on the process state variable.

ALARM Resource - "alarm"

alarm starts a timer of variable duration which will fire at a specified number of clock ticks (each tick is 1/60th of a second) in the future. The **ALARM** resource is implicitly requested when the **alarm** function is called and is ready for service after the timer completes. The **own** function will indicate whether the timer has completed; the **wait** function will wait for the timer to complete if it has not done so. The function call **alarm(0)** cancels the previous call to **alarm**. See the manual page **RESOURCES(3R)** in the *630 MTG Software Reference Manual* for more details.

MSG Resource

The 630 MTG provides for message passing between applications. Each application may have one or more message queues associated with it that other applications can access. If some activity occurs at any message queue associated with the application, a flag is set which indicates the **MSG** resource is ready for service. The **own** and **wait** functions test and wait for, respectively, this condition. For complete details on and examples of the message facility, see the "Interprocess Communication (Messages)" Chapter and the manual pages **RESOURCES(3R)**, **MSGCTL(3L)**, **MSGGET(3L)**, and **MSGOP(3L)** in the *630 MTG Software Reference Manual*.

Example Programs

The source code for the example programs on resources is included in this section (Figures 5-1 through 5-5).

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

#define FW FONTWIDTH(largefont)    /* See Manual Page STRING(3R)*/
#define FH FONTHEIGHT(largefont)  /* See Manual Page STRING(3R)*/

/* List of routines and associated manual page */
Point add();                       /* PTARITH(3R) */
void box();                         /* BOX(3R) */
Point fPt();                       /* FPT(3L) */
Rectangle fRpt();                 /* FPT(3L) */
int eqpt();                       /* EQ(3R) */
void lprintf();                  /* LPRINTF */
void moveto();                   /* MOVETO(3R) */
int own();                       /* RESOURCES(3R)*/
Rectangle raddp();              /* RECTARITH(3R)*/
void rectf();                   /* RECTF(3R) */
unsigned long realtime();       /* REALTIME(3R) */
int request();                  /* RESOURCES(3R)*/
Point sub();                    /* PTARITH(3R) */
Point sPtCurrent();            /* MOVETO(3L) */
int wait();                    /* RESOURCES(3R)*/
```

Figure 5-1: Example Program - "mouse.c" (Sheet 1 of 10)

Example Programs

```
/*Library macros and associated manual pages */
/* int button()          BUTTONS(3R) */
/* int button1()        BUTTONS(3R) */
/* int button2()        BUTTONS(3R) */
/* int button3()        BUTTONS(3R) */
/* int button12()       BUTTONS(3R) */
/* int button13()       BUTTONS(3R) */
/* int button23()       BUTTONS(3R) */
/* int button123()      BUTTONS(3R) */
/* int btn()            BUTTONS(3R) */
/* int btn1()           BUTTONS(3R) */
/* int btn2()           BUTTONS(3R) */
/* int btn3()           BUTTONS(3R) */
/* int btn12()          BUTTONS(3R) */
/* int btn23()          BUTTONS(3R) */
/* int btn13()          BUTTONS(3R) */
/* int btn123()         BUTTONS(3R) */

/* Global Variables in this Program */
/*
** Previous status of button macro calls.
*/
int Prev[4][7];
/*
** Defines the table of button macro names.
*/
Rectangle Table[4][7];

main()
{
    int x,y;                /* general coordinates */
    Point origin, corner;  /* general points */
    Point mxy, mjxy;       /* previous mouse coordinates */
    int Button;            /* return status of button macro */
    unsigned long time;    /* time to update mouse coordinates */
    Point scx, scy, wcx, wcy; /* points to print mouse coord's */
    Rectangle ExitRect;    /* Exit Rectangle */

    /*
    ** Request the mouse resource
    */
    request(MOUSE);
```

Figure 5-1: Example Program - "mouse.c" (Sheet 2 of 10)

```
/*
** Draw The Exit Rectangle. When button
** one of the mouse is clicked in this
** Rectangle, the program will exit.
*/
ExitRect.origin = sub(Direct.corner, fPt(23*FW, 5*FH));
ExitRect.corner = Direct.corner;
box(&display, ExitRect, F_XOR);

/*
** Print message in ExitRect.
*/
moveto(add(ExitRect.origin, Pt(FW, FH)));
lprintf("Click Button One");
moveto(add(ExitRect.origin, Pt(FW, 2*FH)));
lprintf("In This Box");
moveto(add(ExitRect.origin, Pt(FW, 3*FH)));
lprintf("To Exit.");

/*
** Initialize Table. The Rectangles in this table define where the
** names of each button macro will be positioned in the window.
** Each rectangle is 15 "character widths" wide one "character height"
** high. The first row of the table is on line 6 of the window.
*/
for (x=0; x<4; x++) {
    for (y=0; y<7; y++) {
        origin.x = FW*(15*x + 1);
        origin.y = FH*(y + 6);
        corner.x = origin.x + 15*FW;
        corner.y = origin.y + FH;
        Table[x][y] = fRpt(origin, corner);
    }
}

/*
** Print Title line and label for mouse coordinates.
*/
moveto(Pt(0, 0));
lprintf("0");
lprintf(" MOUSE TRACKING AND BUTTON STATUS0);
lprintf(" Screen Coordinates    Window Coordinates0);
```

Figure 5-1: Example Program - "mouse.c" (Sheet 3 of 10)

Example Programs

```
/*
** Obtain the points to print the mouse coordinate info.
*/
lprintf(" ");
/* screen coordinate x */
scx = sPtCurrent();
lprintf ( "                ");
/* window coordinate x */

wcx = sPtCurrent();
lprintf("0");
/* screen coordinate y */
scy = sPtCurrent();
lprintf ( "                ");
/* window coordinate y */
wcy = sPtCurrent();

/*
** Print the table of button macro names.
** Column 1
*/
moveto(add(Direct.origin, Table[0][0].origin));
lprintf("btttn1()          ");

moveto(add(Direct.origin, Table[0][1].origin));
lprintf("btttn2()          ");

moveto(add(Direct.origin, Table[0][2].origin));
lprintf("btttn3()          ");

moveto(add(Direct.origin, Table[0][3].origin));
lprintf("btttn12()         ");

moveto(add(Direct.origin, Table[0][4].origin));
lprintf("btttn13()         ");

moveto(add(Direct.origin, Table[0][5].origin));
lprintf("btttn23()         ");

moveto(add(Direct.origin, Table[0][6].origin));
lprintf("btttn123()        ");
```

Figure 5-1: Example Program - "mouse.c" (Sheet 4 of 10)

```
/*
** Column 2
*/
moveto(add(Direct.origin, Table[1][0].origin));
lprintf("button1()      ");

moveto(add(Direct.origin, Table[1][1].origin));
lprintf("button2()      ");

moveto(add(Direct.origin, Table[1][2].origin));
lprintf("button3()      ");

moveto(add(Direct.origin, Table[1][3].origin));
lprintf("button12()     ");

moveto(add(Direct.origin, Table[1][4].origin));
lprintf("button13()     ");

moveto(add(Direct.origin, Table[1][5].origin));
lprintf("button23()     ");

moveto(add(Direct.origin, Table[1][6].origin));
lprintf("button123()    ");

/*
** Column 3
*/
moveto(add(Direct.origin, Table[2][0].origin));
lprintf("btttn(1)       ");

moveto(add(Direct.origin, Table[2][1].origin));
lprintf("btttn(2)       ");

moveto(add(Direct.origin, Table[2][2].origin));
lprintf("btttn(3)       ");
```

Figure 5-1: Example Program - "mouse.c" (Sheet 5 of 10)

Example Programs

```
/*
** Column 4
*/
moveto(add(Direct.origin, Table[3][0].origin));
lprintf("button(1)      ");

moveto(add(Direct.origin, Table[3][1].origin));
lprintf("button(2)      ");

moveto(add(Direct.origin, Table[3][2].origin));
lprintf("button(3)      ");

/*
** Main loop of the Program.
*/
for(;;) {
    /*
    ** If we own the mouse
    */
    if(own(&MOUSE) {

        /*
        ** Update the mouse coordinates 5 times per second.
        */
        if( realtime() > time) {
            time = realtime() + 12;
            if(!eqpt(mouse.xy, mxy) {
                mxy = mouse.xy;
                moveto(scx);
                /*
                ** Erase old coordinate
                */
                lprintf("      ");
                moveto(scx);
                /*
                ** Draw new coordinate.
                */
                lprintf("x=%d", mouse.xy.x);
                moveto(scy);
```

Figure 5-1: Example Program - "mouse.c" (Sheet 6 of 10)

```
        /*
        ** Erase old coordinate
        */
        lprintf("      ");
        moveto(scx);
        /*
        ** Draw new coordinate.
        */
        lprintf("y=%d", mouse.xy.y);
    }

    if(!eqpt(mouse.jxy, mjxy)) {
        mjxy = mouse.jxy;
        moveto(wcx);
        /*
        ** Erase old coordinate
        */
        lprintf("      ");
        moveto(wcx);
        /*
        ** Draw new coordinate.
        */
        lprintf("x=%d",mouse.jxy.x);
        moveto(wcy);
        /*
        ** Erase old coordinate
        */
        lprintf("      ");
        moveto(wcy);
        /*
        ** Draw new coordinate.
        */
        lprintf("y=%d",mouse.jxy.y);
    }
}
```

Figure 5-1: Example Program - "mouse.c" (Sheet 7 of 10)

Example Programs

```
/*
** The following code calls the button macros and
** determines whether the corresponding name should
** should be high-lighted in the table.
*/

/*
** Column 1 of Table.
*/
if ((Button = btn1()) != Prev[0][0]) {
    Prev[0][0] = Button;
    rectf(&display, raddp(Table[0][0], Drect.origin), F_XOR);
}
if ((Button = btn2()) != Prev[0][1]) {
    Prev[0][1] = Button;
    rectf(&display, raddp(Table[0][1], Drect.origin), F_XOR);
}

if ((Button = btn3()) != Prev[0][2]) {
    Prev[0][2] = Button;
    rectf(&display, raddp(Table[0][2], Drect.origin), F_XOR);
}
Button = btn12();
if (( Button # Prev[0][3]) && (!Button # !Prev[0][3])) {
    Prev[0][3] = Button;
    rectf(&display, raddp(Table[0][3], Drect.origin), F_XOR);
}
Button = btn13();
if (( Button # Prev[0][4]) && (!Button # !Prev[0][4])) {
    Prev[0][4] = Button;
    rectf(&display, raddp(Table[0][4], Drect.origin), F_XOR);
}
Button = btn23();
if (( Button # Prev[0][5]) && (!Button # !Prev[0][5])) {
    Prev[0][5] = Button;
    rectf(&display, raddp(Table[0][5], Drect.origin), F_XOR);
}
Button = btn123();
if (( Button # Prev[0][6]) && (!Button # !Prev[0][6])) {
    Prev[0][6] = Button;
    rectf(&display, raddp(Table[0][6], Drect.origin), F_XOR);
}
}
```

Figure 5-1: Example Program - "mouse.c" (Sheet 8 of 10)

```

/*
** Column 2 of Table.
*/
if ((Button = button1()) != Prev[1][0]) {
    Prev[1][0] = Button;
    rectf(&display, raddp(Table[1][0], Direct.origin), F_XOR);
}
if ((Button = button2()) != Prev[1][1]) {
    Prev[1][1] = Button;
    rectf(&display, raddp(Table[1][1], Direct.origin), F_XOR);
}
if ((Button = button3()) != Prev[1][2]) {
    Prev[1][2] = Button;
    rectf(&display, raddp(Table[1][2], Direct.origin), F_XOR);
}
Button = button12();
if ((Button # Prev[1][3]) && (!Button # !Prev[1][3])) {
    Prev[1][3] = Button;
    rectf(&display, raddp(Table[1][3], Direct.origin), F_XOR);
}
Button = button13();
if ((Button # Prev[1][4]) && (!Button # !Prev[1][4])) {
    Prev[1][4] = Button;
    rectf(&display, raddp(Table[1][4], Direct.origin), F_XOR);
}
Button = button23();
if ((Button # Prev[1][5]) && (!Button # !Prev[1][5])) {
    Prev[1][5] = Button;
    rectf(&display, raddp(Table[1][5], Direct.origin), F_XOR);
}
Button = button123();
if ((Button # Prev[1][6]) && (!Button # !Prev[1][6])) {
    Prev[1][6] = Button;
    rectf(&display, raddp(Table[1][6], Direct.origin), F_XOR);
}

/*
** Column 3 of Table.
*/
if ((Button = btnn(1)) != Prev[2][0]) {
    Prev[2][0] = Button;
    rectf(&display, raddp(Table[2][0], Direct.origin), F_XOR);
}

```

Figure 5-1: Example Program - "mouse.c" (Sheet 9 of 10)

Example Programs

```
if ((Button = btnn(2)) != Prev[2][1]) {
    Prev[2][1] = Button;
    rectf(&display, raddp(Table[2][1], Drect.origin), F_XOR);
}
if ((Button = btnn(3)) != Prev[2][2]) {
    Prev[2][2] = Button;
    rectf(&display, raddp(Table[2][2], Drect.origin), F_XOR);
}

/*
** Column 4 of Table.
*/
if ((Button = button(1)) != Prev[3][0]) {
    Prev[3][0] = Button;
    rectf(&display, raddp(Table[3][0], Drect.origin), F_XOR);
}
if ((Button = button(2)) != Prev[3][1]) {
    Prev[3][1] = Button;
    rectf(&display, raddp(Table[3][1], Drect.origin), F_XOR);
}
if ((Button = button(3)) != Prev[3][2]) {
    Prev[3][2] = Button;
    rectf(&display, raddp(Table[3][2], Drect.origin), F_XOR);
}

/*
** Check for exit.
*/
if( btnn1() && ptinrect(mouse.xy, ExitRect))
    exit();
} else {
    /*
    ** Window is not current.
    ** wait for mouse
    */
    wait(MOUSE);
}
}
}
```

Figure 5-1: Example Program - "mouse.c" (Sheet 10 of 10)

```
#include <dmd.h>
#include "world.h"

/* Library Routines and associated manual page. */
void bitblt();      /* BITBLT(3R)          */
int request();      /* RESOURCES(3R)       */
void sleep();       /* SLEEP(3R)          */

main()
{
    Point MousePosition;

    /*
    ** Request the use of the MOUSE resource.
    */
    request(MOUSE);

    /*
    ** Allow the 630 MTG control process to run
    ** and update the mouse position.
    */
    sleep(2);

    /*
    ** Record the current mouse position.
    */
    MousePosition = mouse.xy;
}
```

Figure 5-2: Example Program "TrackMouse.c" (Sheet 1 of 2)

Example Programs

```
/*
** Draw the world Bitmap at the current
** mouse position.
*/
bitblt(&world, world.rect, &display,
      MousePosition, F_XOR);
for(;;) {
    /*
    ** Erase the world Bitmap from the old
    ** mouse position.
    */
    bitblt(&world, world.rect, &display, MousePosition, F_XOR);
    /*
    ** Update the MousePosition.
    */
    MousePosition = mouse.xy;

    /*
    ** Draw the world at the new position.
    */
    bitblt(&world, world.rect, &display, MousePosition, F_XOR);

    /*
    ** Sleep for two ticks of the 60Hz clock to
    ** release the CPU and synchronize with the
    ** 60Hz refresh rate of the 630 MTG screen.
    */
    sleep(2);
}
}
```

Figure 5-2: Example Program "TrackMouse.c" (Sheet 2 of 2)

```
#include <dmd.h>

/*Library routines and associated manual pages. */
Point add();          /* PTARITH(3R)          */
void exit();         /* EXIT(3R)           */
void jmoveto();      /* JMOVE(3L)          */
void jlineto();     /* JSEGMENT(3L)       */
void jrectf();      /* JRECTF(3L)         */
int kbdchar();      /* KBDCHAR(3R)        */
int local();        /* LOCAL(3R)          */
void nap();         /* SLEEP(3R)          */
Point mul();        /* PTARITH(3R)        */
int request();     /* RESOURCES(3R)      */
void sleep();      /* SLEEP(3R)          */
Point sub();       /* PTARITH(3R)        */
int wait();        /* RESOURCES(3R)      */

/*Library macros and associated manual pages. */
/*int bttn1()      /* BUTTONS(3R)        */
/*int button1()   /* BUTTONS(3R)        */
/*int button2()   /* BUTTONS(3R)        */
/*int button3()   /* BUTTONS(3R)        */

/* Routines local to star.c */
void draw();
void GetDelta();
void GetPoints();
int WhichQuadrant();

#define QUADRANT1 1
#define QUADRANT2 2
#define QUADRANT3 3
#define QUADRANT4 4

typedef struct QuadrantPoints {
    Point quadrant1, quadrant2, quadrant3, quadrant4;
} QuadrantPoints;

QuadrantPoints InitialPoint;
QuadrantPoints FinalPoint;
```

Figure 5-3: Example Program "star.c" (Sheet 1 of 6)

Example Programs

Point MousePosition;

Point Delta;

Point WindowCenter = {XMAX/2, YMAX/2};

/*

** star.c takes an initial point selected by depressing mouse button
** 1, maps that initial point into the four quadrants of the window
** (see the routine WhichQuadrant and SetPoints) and then while
** mouse button 1 is depressed, tracks mouse movement by drawing
** line segments from the initial points in each quadrant to the
** corresponding final points in each quadrant determined by mapping
** the current mouse position into each quadrant. The result is a
** drawing that is symmetrical about the x and y axis. Releasing and
** then depressing mouse button 1 will allow you to select new initial
** points. Mouse button 2 will erase the current drawing.
** star.c exits when the user types a 'q'.

*/

main()

{

/*

** Release the host connection.

*/

local();

/*

** Request the use of the mouse and

** keyboard application resources.

*/

request(MOUSEKBD);

/*

** Main loop.

*/

for(;;) {

/*

** Release the CPU until the mouse or

** the keyboard need service.

*/

wait(MOUSEKBD);

Figure 5-3: Example Program "star.c" (Sheet 2 of 6)

```
if (button1()) {
    /*
    ** If mouse button 1 is depressed,
    ** set initial points in each quadrant.
    */
    GetPoints(&InitialPoint);
    for (;bttn1();) {
        /*
        ** As long as button 1 stays depressed,
        ** get final points in each quadrant
        ** and draw the line segments.
        */
        GetPoints(&FinalPoint);
        draw();
        /*
        ** Busy loop for two clock ticks
        ** in order to allow the user to
        ** move the mouse.
        */
        nap(2);
    }
} else if (button2())
    /*
    ** If mouse button two is depressed
    ** erase the current drawing.
    */
    jrectf(Jrect, F_CLR);
else if (button3()) {
    /*
    ** If buttons 3 is depressed, release
    ** the Mouse resource and allow the control
    ** process to run.
    */
    request(KBD);
    sleep(2);
    request(MOUSE|KBD);
}
}
```

Figure 5-3: Example Program "star.c" (Sheet 3 of 6)

Example Programs

```
        if (kbdchar() == 'q')
            /*
             ** If the user types a 'q', then exit.
             */
            exit();
    }
}

int
WhichQuadrant()
/*
** star.c divides the window into four equal quadrants.
**
**      2      |      1
**      |      |
** -----> x
**      |      |
**      3      |      4
**      |      |
**      v
**      y
** WhichQuadrant will return the number corresponding to
** the quadrant that the mouse cursor is currently in.
**/
{
    if (MousePosition.x >= XMAX/2) {
        if (MousePosition.y <= YMAX/2)
            return(QUADRANT1);
        else
            return(QUADRANT4);
    } else {
        if (MousePosition.y <= YMAX/2)
            return(QUADRANT2);
        else
            return(QUADRANT3);
    }
}

void
GetDelta()
```

Figure 5-3: Example Program "star.c" (Sheet 4 of 6)

```
/*
** The x and y coordinate of the Point Delta helps in
** mapping the current mouse position into the four
** quadrants.
** Delta.x = 2*((XMAX/2) - MousePosition.x)
** Delta.y = 2*((YMAX/2) - MousePosition.y)
*/
{
    MousePosition = mouse.jxy;
    Delta = mul( sub(WindowCenter, MousePosition), 2);
}

void
GetPoints(Qp)
struct QuadrantPoints *Qp;
/*
** Map the current mouse position into the four quadrants.
*/
{
    Point p1, p2, p3, p4;

    GetDelta();
    p1 = MousePosition;
    p2 = MousePosition; p2.x += Delta.x;
    p3 = add(MousePosition, Delta);
    p4 = MousePosition; p4.y += Delta.y;
    switch(WhichQuadrant()) {
    case QUADRANT1:
        Qp->quadrant1 = p1;
        Qp->quadrant2 = p2;
        Qp->quadrant3 = p3;
        Qp->quadrant4 = p4;
        break;
    case QUADRANT2:
        Qp->quadrant1 = p2;
        Qp->quadrant2 = p1;
        Qp->quadrant3 = p4;
        Qp->quadrant4 = p3;
        break;
    }
```

Figure 5-3: Example Program "star.c" (Sheet 5 of 6)

Example Programs

```
    case QUADRANT3:
        Qp->quadrant1 = p3;
        Qp->quadrant2 = p4;
        Qp->quadrant3 = p1;
        Qp->quadrant4 = p2;
        break;
    case QUADRANT4:
        Qp->quadrant1 = p4;
        Qp->quadrant2 = p3;
        Qp->quadrant3 = p2;
        Qp->quadrant4 = p1;
        break;
}

void
draw()
/*
** Draw line segments from the initial point in the quadrant
** to the final point.
*/
{
    int i;
    for (i=0;i<4;i++) {
        jmoveto(((Point *)&InitialPoint)[i]);
        jlineto(((Point *)&FinalPoint)[i], F_OR);
    }
}
```

Figure 5-3: Example Program "star.c" (Sheet 6 of 6)

```
#include <dmd.h>

#define NEWLINE 0x0a
#define RETURN 0x0d

/* Library Routines and associated manual page. */
void exit();          /* EXIT(3R)          */
int kbdchar();        /* KBDCHAR(3R)       */
void lprintf();       /* PRINTF(3R)        */
void lputchar();      /* LPUTCHAR(3L);     */
int psendchar();      /* PSENDCHAR(3R)     */
int request();        /* RESOURCES(3R)     */
int wait();           /* RESOURCES(3R)     */

main()
{
    char c;
    int resources;

    /*
    ** Request the use of the keyboard
    ** and the printer.
    */
    resources = request(KBDPSEND);

    /*
    ** If the request for the printer resource failed,
    ** ask the user if he wants to try the request
    ** again. If not then exit.
    */
    while (!(resources & PSEND)) {
        lprintf("\n Printer Not Available.\n");
        lprintf(" Shall I try again? (y/n)\n");

        while ((c=kbdchar()) == -1) wait(KBD);
        if(c == 'y' || c == 'Y')
            resources = request(KBDPSEND);
        else
            exit();
    }
}
```

Figure 5-4: Example Program "type.c" (Sheet 1 of 2)

Example Programs

```
lprintf(" Type @ to quit.\n");
/*
** Send all characters typed to the printer.
*/
for(;;) {
    while ((c = kbdchar()) == -1) wait(KBD);
    if (c == '@') {
        exit();
    } else {
        /*
        ** Echo character on the screen
        ** and then send it to the printer.
        */
        lputchar(c);
        psendchar(c);
        if (c == RETURN) {
            lputchar(NEWLINE);
            psendchar(NEWLINE);
        }
    }
}
```

Figure 5-4: Example Program "type.c" (Sheet 2 of 2)

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

/* Library Routines and associated manual page. */
Point add();           /* PTARITH(3R)           */
Point fPt();          /* FPT(3L)             */
void lputchar();      /* LPUTCHAR(3L)        */
void moveto();        /* MOVETO(3L)          */
int own();            /* RESOURCES(3R)       */
void rectf();         /* RECTF(3R)           */
int request();        /* RESOURCES(3R)       */
void ringbell();     /* RINGBELL(3R)        */
Point sPtCurrent();  /* MOVETO(3L)          */
int wait();           /* RESOURCES(3R)       */

/* Library Macros and associated manual page. */
/* int FONTWIDTH      STRING(3R)           */
/* int FONTHEIGHT     STRING(3R)           */

/* Local routines in this file */
void init_screen();
void cursor();
void dispchar();

main()
{
    /*
     ** Initialize the current screen point and
     ** draw a cursor.
     */
    init_screen();

    /*
     ** Request the RCV resource.
     */
    request(RCV);
}
```

Figure 5-5: Example Program "vstern1.c" (Sheet 1 of 4)

Example Programs

```
/*
** Main loop of program.
*/
for (;;) {
    /*
    ** Wait for chars from host.
    */
    wait(RCV);

    if (own() & RCV) {
        /*
        ** Erase the current cursor.
        */
        cursor();

        /*
        ** Display characters received.
        */
        dispchar();

        /*
        ** Redraw cursor.
        */
        cursor();
    }
}

void
init_screen()
{
    Point p;

    p = Direct.origin;
    p.y += 3;

    /*
    ** Initialize the current screen point.
    */
    moveto(p);
    cursor();
}
```

Figure 5-5: Example Program "vstern1.c" (Sheet 2 of 4)

```
void
cursor()
{
    /*
    ** This routine is used to draw and erase the cursor.
    */
    Rectangle r;
    extern Point sPtCurrent();
    extern Point add();
    extern Point fPt();

    /*
    ** Set dimension and position of cursor Rectangle.
    */
    r.origin = sPtCurrent();
    r.corner = add(r.origin, fPt(FONTWIDTH(mediumfont),
                                FONTHEIGHT(mediumfont)));

    /*
    ** the following rectf will erase the cursor if it already exists
    ** and draw it if it does not.
    */
    rectf(&display, r, F_XOR);
}

void
dispchar()
{
    register int c;
    Point curpos;

    /*
    ** Process all characters received.
    */
}
```

Figure 5-5: Example Program "vstern1.c" (Sheet 3 of 4)

Example Programs

```
while ( own() & RCV ) {
    switch(c = rcvchar()) {
        case '\007':
            ringbell();
            break;

        case '\b' :    /* backspace */
            curpos = sPtCurrent();
            if(curpos.x - FONTWIDTH(largefont) >= Drect.origin.x)
                curpos.x -= FONTWIDTH(largefont);
            moveto(curpos);
            break;

        default:
            lputchar(c);
            break;
    }
}
```

Figure 5-5: Example Program "vsterm1.c" (Sheet 4 of 4)

Chapter 6: User Interface Toolbox

Introduction	6-1
“menuhit”	6-2
Data Types - “Menu”	6-2
Mouse Interaction	6-4
Using “menuhit”	6-5
Generating a Simple Menu - “Menu1.c”	6-5
Generating Multiple Menus - “Menu2.c”	6-5
“tmenuhit” - Tree Menu	6-6
“tmenuhit” Data Types - “Tmenu” and “Titem”	6-6
“Tmenu” Definition	6-7
“Titem” Definition	6-8
Calling “tmenuhit”	6-9
“tmenuhit” Return Value	6-10
Using “tmenuhit”	6-10
Customizing the “Titem” Data Type for Simple Menus - “Tmenu1.c”	6-10
Using the “next” Field to Generate Submenus - “Tmenu2.c”	6-11
Menu Expansion - “Tmenu3.c”	6-11
Menu Item Bitmaps - “Tmenu4.c”	6-12
Multiple Fonts - “Tmenu5.c”	6-12
Menu Item Greying and the “hfn” Subroutine - “Tmenu6.c”	6-13
Static Menus - “Tmenu7.c”	6-13
Static Menus and Multiple Selections - “Tmenu8.c”	6-14
“dfn”, “hfn”, “bfn” Demonstration - “Tmenu9.c”	6-14
The Label Bar	6-15
Structure of Label Bar	6-15
Requesting the Label Bar - “labelon” and “labeloff”	6-16
Displaying Icons - “labelicon”	6-16
Displaying Text - “labeltext”	6-17
Displaying Text and Icons	6-17
Maintaining Text and Icons After Reshape	6-18

Chapter 6: User Interface Toolbox

Message Boxes 6-19

Example Programs 6-21

Introduction

menuhit, **tmenuhit**, label bars, and message boxes are four tools that enhance the user interface of an application program. **menuhit** enables an application program to display a menu in response to depressing a mouse button. From the menu, a user can make a selection by releasing the button over a desired selection. **tmenuhit**, an advanced version of **menuhit**, allows multiple menus to be presented simultaneously in a hierarchical structure or tree. The label bar is a bar that is displayed across the top of a window. By using icons and text strings, information about the application's current status and its resources can be printed within the label bar. The message box is a friendly interface for providing information to the user through pop-up boxes.

Several example programs have been included in this chapter to demonstrate the different user interface tools. The source code for these programs is in directory *\$DMD/examples/UserInterface*. Also, a printout of the source code for each program is included at the end of this chapter in the "Example Programs" section.

“menuhit”

menuhit is declared as:

```
int menuhit (m, n)

Menu *m;
int n;
```

The **menuhit** routine accepts the two parameters "m" and "n". The "m" parameter is a pointer to the **Menu** data structure. The "n" parameter indicates which mouse button is to be used for interaction with the user.

When invoked, **menuhit** displays the menu, specified by the **Menu** data structure that is pointed to by "m", and waits for the user to make a selection. Once a menu item is selected, **menuhit** returns an integer indicating the selection. See the manual page on **MENUHIT(3L)** for more details.

Data Types - “Menu”

The **Menu** data structure is defined in the include file **dmd.h** as:

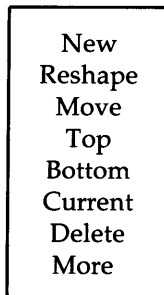
```
typedef struct Menu {
    char **item;
    short prevhit;
    short prevtop;
    char *(*generator)();
} Menu;
```

The **Menu** data fields are defined as follows:

item This is a pointer to an array of character strings. The character strings in this array, called menu items, are displayed within a rectangular box when **menuhit** is called. For example, the following array of character strings:

```
char *MenuItems[] = {
    "New",
    "Reshape",
    "Move",
    "Top",
    "Bottom",
    "Current",
    "Delete",
    "More",
    (char *)0,
};
```

would be displayed in a menu as:



Note the last entry in the array is a null pointer which allows **menuhit** to detect the end of the array.

prevhit This is an integer that is maintained by **menuhit** to designate the item selected from the previous call to **menuhit**. When **menuhit** is called, it will try to position the mouse cursor over the previously selected item.

prevtop This is an integer that is maintained by **menuhit** to indicate the top menu item from the previous call to **menuhit**. **prevtop** is used when there are more than 16 menu items to be displayed in a

“**menuhit**”

scrolling menu. Details on scrolling menus and an example can be found in the manual page **MENUHIT(3L)** in the *630 MTG Software Reference Manual*.

generator

This is a pointer to a function that **menuhit** will call if **item** is zero. The **generator** function, which must be defined within the application program, dynamically generates the items to be displayed in a menu. An example of a menu using a **generator** function is given in the manual page **MENUHIT(3L)**.

Mouse Interaction

The mouse button for bringing up menus is specified as a parameter to the **menuhit** routine. When the mouse is set up for right-hand operation, the button numbering is as follows:

- 1 — left button
- 2 — middle button
- 3 — right button.

When the mouse is set up for left-hand operation, the button numbering is as follows:

- 1 — right button
- 2 — middle button
- 3 — left button.

Refer to the *630 MTG Terminal User's Guide* for more information on terminal setup.

menuhit assumes that the specified button is depressed when it is called. When the user releases the mouse button, **menuhit** will return an integer indicating the user's selection. The integer returned is an index into the array of character strings pointed to by **item**.

Using “menuhit”

The **Menu1.c** and **Menu2.c** example programs in the directory *\$DMD/examples/UserInterface* demonstrate the **menuhit** routine. These programs, as all others in this chapter, should be compiled and downloaded using commands similar to the following for each program:

```
dmdcc -o Menu1 Menu1.c
dmdld Menu1
```

Generating a Simple Menu - “Menu1.c”

Menu1.c is a simple demonstration of the use of **menuhit**. Once downloaded, depressing button 2 brings up a menu displaying "Breakfast", "Lunch", and "Dinner" as menu items. Moving the cursor over the desired item and releasing button 2 selects the item. Clicking button 1 causes the program to exit. A printout of the source code for **Menu1.c** is given in Figure 6-1 at the end of this chapter.

Generating Multiple Menus - “Menu2.c”

Menu2.c expands **Menu1.c** by allowing the user to bring up a submenu in response to a main menu selection. Each submenu allows the user to make additional selections and to return to the main menu. A printout of **Menu2.c** is given in Figure 6-2.

“tmenuhit” - Tree Menus

tmenuhit, an enhanced version of **menuhit**, includes the following features:

- Display of multiple menus in a hierarchical or tree format
- Programmable menu item identification field
- Menu item greying for non-selectable menu items
- Use of Bitmaps within menu items
- Font selection for menu item text
- Execution of user-defined subroutines within **tmenuhit**
- "Static" menus
- Programmable positioning of menus on screen
- Multiple menu item selections during one call to **tmenuhit**
- Programmable selection of the above features.

tmenuhit will present the user with one or more menus in a hierarchical format, allow the user to make a selection using the mouse buttons, and return a pointer to a **Titem** data structure. See the manual page **TMENUHIT(3R)** for more details.

“tmenuhit” Data Types - “Tmenu” and “Titem”

The **Tmenu** data type is very similar to the **Menu** data type for **menuhit** with differences being in the definition of the **item** field, the return value of the **generator** function, and the addition of a new field called **menumap**.

“Tmenu” Definition

The **Tmenu** data type is declared within the include file **menu.h** as follows:

```
typedef struct Tmenu
{
    Titem    *item;
    short    prevhit;
    short    prevtop;
    Titem    *(*generator)();
    short    menumap;
} Tmenu;
```

The definitions of the fields in the **Tmenu** data structure are as follows:

- item** This is a pointer to an array of **Titem** data structures. A **Titem** defines a single menu item. The array of **Titems** defines all the menu items in a single menu. The last **Titem** in the array must have its **text** field set to **(char *)0**.
- prevhit** This is an integer maintained by **tmenuhit** and used to designate the menu item selected from the previous call to **tmenuhit**. When **tmenuhit** is called, it tries to position the mouse cursor over the previously selected menu item if possible.
- prevtop** This is an integer maintained by **tmenuhit** and used to indicate the top menu item from the previous call to **tmenuhit**. **tmenuhit** uses **prevtop** when there are more than 16 menu items to be displayed in a scrolling menu.
- generator** This is a pointer to a function (defined in the application program) that **tmenuhit** will call if the **item** field in the **Tmenu** data structure is set to zero. The **generator** function must dynamically generate and return pointers to the **Titem** data structures needed to create a menu. **tmenuhit** will call the **generator** function repeatedly until it returns a **Titem** with its **text** field equal to **(char *)0**.
- menumap** This is a bit vector that allows you to tailor the **Titem** data type for each instance of a menu. This enables you to select and initialize only those **Titem** data fields needed in your application.

“Titem” Definition

Each menu is composed of an array of **Titem** data structures. Each **Titem** in the array defines a single menu item in the menu. The **Titem** data type is defined in **menu.h** as follows:

```
typedef struct Titem
{
    char          *text;
    struct {
        unsigned short  uval;
        unsigned short  grey;
    } ufield;
    struct Tmenu   *next;
    Bitmap         *icon;
    struct Font    *font;
    void          (*dfn)();
    void          (*bfm)();
    void          (*hfn)();
} Titem;
```

The definitions of the fields in **Titem** data structure are as follows:

- text** This is a pointer to a NULL terminated character string to be displayed in the menu for this item. Customized **Titem** data structures must always include the **text** field.
- uval** This is an integer typically used as a menu item identification field containing a constant that uniquely identifies the **Titem**. This is useful since **tmenuhit** does not return an integer index, as does **menuhit**, but returns a pointer to a **Titem** structure that corresponds to the selected menu item.
- grey** This is an integer that, if set to one by the application program, will cause **tmenuhit** to display the menu item with a "textured" background and make the menu item non-selectable. A null pointer will be returned by **tmenuhit** if a "greyed" menu item is selected.
- next** This points to a **Tmenu** data structure which defines a submenu associated with this **Titem**. When a submenu is available for a menu item, an arrow icon is placed to the left of the menu item text. The menu containing the menu item that has a submenu is

called the "parent" menu. A "parent" menu can have as many submenus as it has menu items. The topmost parent menu is called the "root" menu.

icon This is a pointer to a Bitmap. The Bitmap will be displayed to the left of menu item text. The Bitmaps within different menu items of the same menu can have different sizes.

dfn, bfn, hfn

These are pointers to functions defined within your application program. These functions, if initialized, will be called by **tmenuhit**. **dfn** is called before sliding "down" into a submenu associated with a **Titem**. If there is no submenu, **dfn** will never be called. **bfn** is called after sliding from a submenu "back" to the parent menu. **hfn** is called upon selecting an item (a "hit").

Calling “tmenuhit”

tmenuhit is called with three or possibly four parameters, depending on the bits set in the **flags** parameter. **tmenuhit** is declared as follows:

```
Titem *tmenuhit(m, n, flags [,p])
```

```
Tmenu *m;  
int n;  
int flags;  
Pointp;
```

The definitions for the **tmenuhit** parameters are as follows:

- m** This is a pointer to the root **Tmenu** data structure. Although multiple **Tmenu** data structures comprise a hierarchical menu, **tmenuhit** only requires the address of the root **Tmenu**. The fact that submenus exist has no effect on the call to **tmenuhit**.
- n** This specifies the mouse button that **tmenuhit** will use for interaction with the user. If this parameter equals 0, **tmenuhit** will use all the mouse buttons.
- flags** This is a bit vector that gives the user some control over the operation of **tmenuhit**.

- p** This is an optional parameter that specifies a point for the upper left corner of the root menu. (The "p" parameter will be discussed and demonstrated in the example program **Tmenu8.c**.)

“tmenuhit” Return Value

tmenuhit returns a pointer to a **Titem** as defined in **menu.h**. In order to use the returned pointer with customized **Titems**, the return value must be type cast to the user's customized **Titem**. Properly type casting the return value of **tmenuhit** can become confusing when there are too many customized **Titem** data types. Therefore, the user should determine a minimum subset of the **Titem** fields required for the application and declare only one customized **Titem**. Examples of customized **Titems** are described in the following section.

Using “tmenuhit”

To become familiar with **tmenuhit**, it is suggested that you compile and run each of the examples discussed in this section. The source code for the examples can be found in the directory `$DMD/examples/UserInterface` and in the printouts at the end of this chapter. Refer to the manual page **TMENUHIT(3R)** in the *630 MTG Software Reference Manual* for details on the **tmenuhit** features used.

Compile and download each program as it is reviewed.

Customizing the “Titem” Data Type for Simple Menus - “Tmenu1.c”

Tmenu1.c demonstrates how **tmenuhit** can be used to generate menus similar to the menus of **menuhit**. The source code for this example is shown in Figure 6-3.

Tmenu1.c is a recoded version of **Menu1.c** using **tmenuhit** to display **MainMenu**. **Tmenu1.c** also allows the user to select a menu item using mouse button 2.

To construct a menu, you must first determine which fields of the **Titem** data type you want to use in your application. In **Tmenu1.c**, only the **text** field was needed, as shown in the type definition of **MainTitem** in **Tmenu1.c**.

The **menumap** field in the **Tmenu** data structure must be set to reflect your customized **Titem**. This is accomplished by setting **menumap** equal to **TM_TEXT**. Refer to the definition **MAP1** and the initialization of **MainMenu** in **Tmenu1.c**.

Note: The **text** field must always be included in your **Titem** because **tmenuhit** looks for the **text** field set to "(char *)0" to locate the end of the **Titem** array.

Using the “next” Field to Generate Submenus - “Tmenu2.c”

Tmenu2.c demonstrates the use of the **next** field in the **Titem** data type to generate submenus. The source code for **Tmenu2.c** is shown in Figure 6-4.

The customized **Titem**, **MyTitem**, in **Tmenu2.c** uses both the **text** and **next** fields. The **next** field is used to generate submenus. The order of the fields declared in any customized **Titem** must be the same as the order of the fields in the **Titem** data type declared in **menu.h**.

Note: Each menu, in a multiple menu display, could use its own customized **Titem** data type. This, however, can become very confusing when there are many **Titem** data types in one application, especially when it comes to properly type casting the return value of **tmenuhit**. It is suggested that you determine a minimum subset of the **Titem** data fields you need in your application and then declare one customized **Titem**.

Linking the Menus Together

In order for a menu item to be linked with a submenu, the **next** field of the **Titem** must point to the **Tmenu** data structure associated with the submenu. This linkage occurs in the **MainItems** array in **Tmenu2.c**. When a menu item does not have a submenu, **next** must be set to zero.

Menu Expansion - “Tmenu3.c”

There is a slight change to the **tmenuhit** call in **Tmenu3.c**. Instead of passing the **flags** parameter as a zero, it is set to **TM_EXPAND** as shown below:

```
item = (MyTitem *)tmenuhit(&MainMenu, BUTTON2, TM_EXPAND);
```

“tmenuhit” - Tree Menus

This change instructs **tmenuhit** to expand the menu tree down to the submenu that contains the menu item selected in the previous call to **tmenuhit**. The source code for **Tmenu3.c** is shown in Figure 6-5.

Menu Item Bitmaps - “Tmenu4.c”

Tmenu4.c demonstrates the use of Bitmaps with menu items. The Bitmaps needed for this example are in the file:

```
$DMD/examples/UserInterface/MenuIcons.h.
```

MyTitem now includes the data field **icon** which is a pointer to a Bitmap. This Bitmap will be displayed to the left of the menu item text. If no Bitmap is to be displayed, **icon** should be set to zero. A printout of **Tmenu4.c** is given in Figure 6-6.

Multiple Fonts - “Tmenu5.c”

Tmenu5.c demonstrates the use of the **font** field in the **Titem** data structure. The source code for **Tmenu5.c** is shown in Figure 6-7.

Using the **font** field in the **Titem** data structure allows you to select a font style to use when displaying the menu item text string. **Tmenu5.c** uses the three 630 MTG resident fonts. Any font style can be used as long as it exists in the terminal’s font cache or was downloaded from the host by your application using one of the font loading routines. See the Chapter "Fonts and the Font Cache" for more details on fonts.

Initialization of the “font” Field

The **font** fields must be dynamically initialized when using the 630 MTG resident fonts. This is because at compile time, the compiler does not know the address in ROM of the resident fonts. Trying to initialize the fonts statically with the rest of the fields of the **MyTitem** data structure will cause the compiler to generate an error about illegal initialization of data. This rule is true when using any of the 630 MTG ROM resident data structures.

Menu Item Greying and the “hfn” Subroutine - “Tmenu6.c”

Tmenu6.c demonstrates two additional features of **tmenuhit**:

- Menu item greying
- User-defined subroutines called from within **tmenuhit**.

The source code for **Tmenu6.c** is shown in Figure 6-8.

Two additional fields, **ufield** and **hfn**, have been added to the customized **Titem** data type, **MyTitem**. The **ufield** data structure is located between the **text** and **next** field in **MyTitem**, and the **hfn** structure is located after the **font** field in **MyTitem**. The addition of these fields is reflected in the **menumap** by adding **TM_UFIELD** and **TM_HFN** to the definition of **MAP1**. The body of **Tmenu6.c** is equivalent to the body of **Tmenu5.c**.

The routine **SetGrey** is the **hfn** function and is called by **tmenuhit** when the user makes a menu item selection. **SetGrey** sets the **grey** field to one in the **ufield** data structure for the selected item. When the **grey** field is set to one, the menu item is displayed with a textured background and is also non-selectable.

Static Menus - “Tmenu7.c”

Tmenu7.c demonstrates the use of the **TM_STATIC** flag. The **TM_STATIC** flag instructs **tmenuhit** to reverse the sense of the mouse buttons in terms of depression and release. To bring up the menu in example **Tmenu7.c**, you must click button 2 in the application’s window. A menu will pop up on the screen but it is not necessary to keep the mouse button depressed to keep the menu on the screen. Menu selections are made with a mouse button depression instead of a mouse button release. **tmenuhit** returns as normal after a selection or non-selection. The source code for **Tmenu7.c** is shown in Figure 6-9.

Static Menus and Multiple Selections - “Tmenu8.c”

Tmenu8.c continues the demonstration of static menus started in **Tmenu7.c**. The source code for **Tmenu8.c** is shown in Figure 6-10.

Tmenu8.c adds two features to **Tmenu7.c**. First of all, **TM_POINT** is set in the **flags** parameter. This allows a fourth parameter, "p", to be passed to **tmenuhit**. "p" specifies a point at which to draw the upper left corner of the root menu.

Secondly, the flag **TM_NORET** is set. This flag instructs **tmenuhit** not to return to the calling application until a non-selection has been made. Therefore, the user is allowed to make multiple selections in a menu without having to bring the menu up for every selection. The menus in this example operate the same way as the static menus used in the 630 MTG terminal Setup.

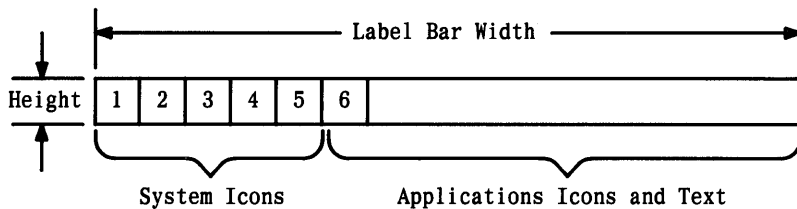
“dfn”, “hfn”, “bfn” Demonstration - “Tmenu9.c”

Tmenu9.c breaks with the previous examples and gives an interesting demonstration of the use of the **dfn**, **hfn**, and **bfn** functions of **tmenuhit**. The source code for **Tmenu9.c** is shown in Figure 6-11.

The Label Bar

Structure of Label Bar

The Label Bar is a rectangle that appears across the top interior of an application's window. It is used by the 630 MTG operating system to display status information about the application and can be used by the application program to display its own status information in the form of text and/or icons. The following is an illustration of the label bar.



The pixel height of the label bar rectangle is defined in the include file **label.h** by the constant, LABEL_HEIGHT.

```
#define LABEL_HEIGHT 20 /* height of label area */
```

The width of the label bar is equal to the interior width of the application's window, which can be expressed as:

```
label_width = Direct.corner.x - Direct.origin.x;
```

The interior of the label bar is divided into two sections: "system icons" and "application icons and text". The "system icons" section is composed of five icon positions. Each icon position is assigned an integer index from one to five and spans an area 16-pixels high and 16-pixels wide. These reserved positions are defined in the include file **label.h** as:

The Label Bar

```
#define L_HOST_POSITION      1 /* current host connection */
#define L_MUX_POSITION      2 /* current host environment*/
#define L_PRINT_POSITION    3 /* printer request status */
#define L_SCROLL_POSITION   4 /* scroll lock key status */
#define L_CAP_POSITION      5 /* caps lock key status */
```

The "application icons and text" section begins at icon position 6. The number of application icon positions available and the maximum text string length depend upon the width of the application's window. The application icons and text positions are defined in the include file **label.h** as:

```
#define L_USER_POSITION     6 /* first user position */
```

Requesting the Label Bar - "labelon" and "labeloff"

The **labelon()** routine puts a label bar at the top interior of the application's window. The global variable **Direct** is changed to the smaller interior window size by decreasing the height of **Direct** by 20 pixels. The system icons of the label bar will be updated automatically by the 630 MTG operating system to indicate the following:

- Application's current host connection
- Type of communications protocol (multiplexed or non-multiplexed)
- Printer request status
- State of the scroll lock and caps lock keys.

The **labeloff()** routine will remove the label bar from the window and restore the initial height of **Direct**. See the manual page LABELON(3R) for more details.

Displaying Icons - "labelicon"

labelicon draws a Bitmap in a specified label bar icon position. The Bitmap is clipped to a height of 16 pixels and a width that is bounded on the left by the icon position index and on the right by the edge of the label bar. The width of the Bitmap can be greater than 16-pixels. The first icon position

index that an application program can use is `L_USER_POSITION`. See the manual page `LABELON(3R)` for more details on `labelicon`.

The example program, `label1.c`, displays the skull and crossbones icon in the label bar at icon position six (`L_USER_POSITION`) and icon position seven (`L_USER_POSITION + 1`). Note that Texture16s must be converted into Bitmaps before they can be displayed in the label bar. `label1.c` is located in the directory `$DMD/examples/UserInterface` and a printout is given in Figure 6-12.

Displaying Text - "labeltext"

`labeltext` displays character strings in the "application icons and text" area. The text string can be displayed with the three following justifications:

- `L_LEFT` This left justifies the text string to `L_USER_POSITION`.
- `L_RIGHT` This right justifies the text string to the right border of the label bar.
- `L_CENTER` This centers the string in the full length of the "application icons and text" area.

`labeltext` uses the 630's **mediumfont** font. Text strings are displayed in the `F_XOR` storage mode; therefore, multiple strings displayed in the label bar will superimpose unless previously written strings are erased by rewriting the same string a second time. If the text string is too long to fit in the "application icons and text" area, it will be clipped off at the right edge of the label bar. The example program `label2.c` demonstrates `labeltext`. A printout of `label2.c` is given in Figure 6-13. See the manual page `LABELON(3R)` for more details on `labeltext`.

Displaying Text and Icons

Icons and text strings can share the "application icons and text" area. However, some care is necessary to prevent interference when displaying both text and icons in the label bar. For example, if an application is to display icons in positions 6 and 7 in addition to left-justified text strings, the text string must contain enough leading space characters so as not to overwrite the two icon positions.

The Label Bar

The number of leading spaces required in a text string is calculated by using the following equation:

$$\text{spaces} = (\text{int})(\text{float}(n * 16)/9 + 0.5);$$

where:

"n" is the number of icon positions being used starting at position '6' (L_USER_POSITION)

"16" is the width in bits of an icon position

"9" is the width in bits of the space character in the **mediumfont** font.

The example program **label3.c** (Figure 6-14) is a demonstration of this requirement. Notice the following line of code in **label3.c**:

```
labeltext ("    left", 8, L_LEFT);
```

The four spaces before "left" are based on the following calculation:

$$\text{spaces} = (\text{int})((2 * 16)/9 + 0.5) = 4$$

Four spaces are needed to pad the string "left" to prevent interference with the two skull and crossbone icons.

Maintaining Text and Icons After Reshape

When an application's window is reshaped, the application is responsible for redrawing any icons or text strings it may have written into the label bar. Refer to the "Application Resources" Chapter for information on refreshing the screen after a reshape.

Refer to the manual page **LABELON(3R)** for additional information on label bars.

Message Boxes

A message box is a "pop-up" box that tracks mouse movement until the user presses any mouse button. The 630 operating system uses message boxes to explain to the user why a particular mouse-driven operation failed or is not possible. For example, the 630 MTG will display a message box when the user attempts to delete the last window to a host or create a new window when the 630 MTG does not have enough memory. (The *630 MTG Terminal User's Guide* lists all of the system-generated message boxes.)

The routine **msgbox** creates message boxes for an application. **msgbox** is called with one or more pointers to character strings with the last argument being (char *)0 to terminate the argument list. Each character string will be drawn on a separate line centered within a box and displayed with the **mediumfont** font. Two example programs, **msgbox1.c** and **msgbox2.c**, are located in the directory *\$DMD/examples/UserInterface* and demonstrate the use of message boxes. Printouts of **msgbox1.c** and **msgbox2.c** are given in Figures 6-15 and 6-16, respectively.

msgbox1.c will display a pop-up message box containing the message:



This is a Message
Box Demonstration

when any mouse button is depressed. Click any mouse button to terminate the display of the message box. The program will terminate when any keyboard character is typed and the message box is not being displayed.

msgbox2.c uses the **menuhit** and **msgbox** routines. When **msgbox2.c** is executed, button 2 will bring up a menu. A different message box is associated with each menu item. For example, if the user selects the menu item, **Message Boxes**, a message box containing the message:



Hey!
Hey!
MESSAGE BOXES

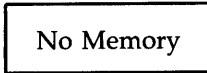
will appear.

Message Boxes

If the first argument to **msgbox** is (char *)0, a message box with the message "No Memory" is displayed. Therefore, if the menu item, **Default**, is selected, the command line:

```
msgbox((char *)0)
```

will produce a message box containing the message:



Refer to the manual page, **MSGBOX(3R)** for additional information on message boxes.

Example Programs

A printout of the source code for the example programs on the user interface toolbox is included in this section (Figures 6-1 through 6-16).

```
#include <dmd.h>

/* Library Routines and associated manual page. */
void exit();           /* EXIT(3R)           */
void lprintf();       /* PRINTF            */
int menuhit();        /* MENUHIT(3L)      */
int request();        /* RESOURCES(3R)    */
void sleep();         /* SLEEP(3R)        */
int wait();           /* RESOURCES(3R)    */

/* Library Macros and associated manual page. */
/*int button1();      BUTTONS(3R)        */
/*int button2();      BUTTONS(3R)        */
/*int button3();      BUTTONS(3R)        */

#define BUTTON2 2

/*
** Declare array of menu items
*/
char *MainItems [] = {
    "Breakfast",
    "Lunch",
    "Dinner",
    (char *)0,
};

Menu MainMenu = { MainItems };
```

Figure 6-1: Example Program "Menu1.c" (Sheet 1 of 2)

Example Programs

```
main()
{
    int m;
    request(MOUSE);
    for(;;) {
        wait(MOUSE);
        if (button1())
            /*
             ** Terminate execution.
             */
            exit();
        else if (button2()) {

            /*
             ** Application menus are normally handled on
             ** button 2. Call "menuhit" to get user's menu
             ** selection and print out selection made.
             */
            m = menuhit(&MainMenu, BUTTON2);
            lprintf("Your selection was %s\n", MainItems[m]);
        } else if(button3()) {
            /*
             ** Release ownership of the mouse and
             ** sleep for two ticks of the 60Hz clock
             ** to let the control process take care of
             ** processing button 3 of the mouse.
             */
            request(0);
            sleep(2);
            /*
             ** Back on the "air," get the mouse back.
             */
            request(MOUSE);
        }
    }
}
```

Figure 6-1: Example Program "Menu1.c" (Sheet 2 of 2)

```
#include <dmd.h>

/* Library Routines and associated manual page. */
void exit();           /* EXIT(3R) */
void lprintf();       /* PRINTF */
int menuhit();        /* MENUHIT(3L) */
int request();        /* RESOURCES(3R) */
void sleep();         /* SLEEP(3R) */
int wait();           /* RESOURCES(3R) */

/* Library Macros and associated manual page. */
/*int button1();      BUTTONS(3R) */
/*int button2();      BUTTONS(3R) */
/*int button3();      BUTTONS(3R) */

#define BUTTON2 2

/*
** Declare main menu.
*/
char *MainItems [] = {
    "Breakfast",
    "Lunch",
    "Dinner",
    (char *)0,
};

/*
** Declare breakfast menu.
*/
char *BreakfastItems [] = {
    "Pancakes",
    "French Toast",
    "Bacon & Eggs",
    "Coffee",
    "Orange Juice",
    "Main Menu",
    (char *)0,
};
```

Figure 6-2: Example Program "Menu2.c" (Sheet 1 of 4)

Example Programs

```
/*
** Declare lunch menu.
*/
char *LunchItems [] = {
    "Hamburger",
    "Hot Dog",
    "French Fries",
    "Root Beer",
    "Grape Soda",
    "Main Menu",
    (char *)0,
};

/*
** Declare dinner menu.
*/
char *DinnerItems [] = {
    "Salad",
    "Steak",
    "Fish",
    "Baked Potato",
    "Wine",
    "Main Menu",
    (char *)0,
};

char **Menus [] = {
    MainItems,
    BreakfastItems,
    LunchItems,
    DinnerItems,
};

main()
{
    int MenuSelection, WhichMenu;
    Menu CurrentMenu;

    WhichMenu = 0;
    CurrentMenu.item = Menus[WhichMenu];
}
```

Figure 6-2: Example Program "Menu2.c" (Sheet 2 of 4)

```
request(MOUSE);
lprintf("\n Press button2 for main menu.");
for(;;) {
    wait(MOUSE);
    if (button1())
        exit();
    else if (button2()) {
        switch(WhichMenu) {
            case 0: /*MainItems*/
                /*
                 ** Display main menu.
                 */
                MenuSelection = menuhit(&CurrentMenu, BUTTON2);
                if (MenuSelection != -1) {
                    /*
                     ** Print main menu selection.
                     */
                    lprintf("\n Press button 2 for %s selections.",
                        MainItems[MenuSelection]);
                    WhichMenu = MenuSelection + 1;
                    CurrentMenu.item = Menus[WhichMenu];
                }
                break;
            case 1: /*BreakfastItems*/
            case 2: /*LunchItems*/
            case 3: /*DinnerItems*/
                /*
                 ** Display submenu.
                 */
                MenuSelection = menuhit(&CurrentMenu, BUTTON2);
                switch(MenuSelection) {
                    case -1:
                        break;
                    case 5: /* Go back to main menu */
                        WhichMenu = 0;
                        CurrentMenu.item = Menus[WhichMenu];
                        lprintf("\n Press button2 for main menu.");
                        break;
                    default:
```

Figure 6-2: Example Program "Menu2.c" (Sheet 3 of 4)

Example Programs

```
        /*
        ** Print submenu selections.
        */
        lprintf("\n \t%s",
                CurrentMenu.item[MenuSelection]);
    }
}
} else if(button3()) {
    /*
    ** Release ownership of the mouse and
    ** sleep for two ticks of the 60Hz clock
    ** to let the control process take care of
    ** processing button 3 of the mouse.
    */
    request(0);
    sleep(2);
    /*
    ** Back on the "air," get the mouse back.
    */
    request(MOUSE);
}
}
}
```

Figure 6-2: Example Program "Menu2.c" (Sheet 4 of 4)

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

/* Library Routines and associated manual page. */
void exit();          /* EXIT(3R)          */
void lprintf();      /* PRINTF           */
int request();       /* RESOURCES(3R)   */
void sleep();        /* SLEEP(3R)       */
Titem *tmenuhit();  /* TMENUHIT(3R)   */
int wait();          /* RESOURCES(3R)   */

/* Library Macros and associated manual page. */
/*int button1();     BUTTONS(3R)       */
/*int button2();     BUTTONS(3R)       */
/*int button3();     BUTTONS(3R)       */

#define BUTTON2 2

/*
** definition for menumap
*/
#define MAP1    TM_TEXT

/*
** definition of customized Titem structure
*/
typedef struct MainTitem {
    char *text;
} MainTitem;

/*
** Initialization of array of Titem1's
*/
MainTitem MainItems [] = {
    "Breakfast",
    "Lunch",
    "Dinner",
    (char *)0,
};
```

Figure 6-3: Example Program "Tmenu1.c" (Sheet 1 of 2)

Example Programs

```
/*
** Initialization of Tmenu structure
*/
Tmenu MainMenu = { (Titem *)MainItems, 0, 0, 0, MAP1 };

main()
{
    MainTitem *item;

    request(MOUSE);
    lprintf("\n Press button 2 to get menu.");
    for (;;) {
        wait(MOUSE);
        if (button1())
            exit();
        else if (button2()) {
            /*
            ** Display menu.
            */
            item = (MainTitem *)tmenuhit(&MainMenu, BUTTON2, 0);
            if (item)
                lprintf("\n Your selection was %s",item->text);
        } else if(button3()) {
            /*
            ** Release ownership of the mouse and
            ** sleep for two ticks of the 60Hz clock
            ** to let the control process take care of
            ** processing button 3 of the mouse.
            */
            request(0);
            sleep(2);
            /*
            ** Back on the "air," get the mouse back.
            */
            request(MOUSE);
        }
    }
}
```

Figure 6-3: Example Program "Tmenu1.c" (Sheet 2 of 2)

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

/* Library Routines and associated manual page. */
void exit();           /* EXIT(3R)           */
void lprintf();       /* PRINTF            */
int request();        /* RESOURCES(3R)    */
void sleep();         /* SLEEP(3R)        */
Titem *tmenuhit();    /* TMENUHIT(3R)     */
int wait();           /* RESOURCES(3R)    */

/* Library Macros and associated manual page. */
/*int button1();      /* BUTTONS(3R)      */
/*int button2();      /* BUTTONS(3R)      */
/*int button3();      /* BUTTONS(3R)      */

#define BUTTON2 2

/*
** definition for menumap
*/
#define MAP1      TM_TEXT | TM_NEXT

/*
** definition of customized Titem structure
*/
typedef struct MyTitem {
    char      *text;
    struct Tmenu      *next;
} MyTitem;

/*
** Initialize array of MyTitem's for Breakfast submenu.
** Menu items do not have submenus
*/
MyTitem BreakfastItems [] = {
    "Pancakes",    0,
    "French Toast", 0,
    "Bacon & Eggs", 0,
    "Coffee",      0,
    "Orange Juice", 0,
    (char *)0,
};
```

Figure 6-4: Example Program "Tmenu2.c" (Sheet 1 of 4)

Example Programs

```
/*
** Initialization of Tmenu structure for Breakfast menu
*/
Tmenu BreakfastMenu = { (Titem *)BreakfastItems, 0, 0, 0, MAP1 };

/*
** Initialize array of MyTitem's for Lunch submenu.
** Menu items do not have submenus.
*/
MyTitem LunchItems [] = {
    "Hamburger",    0,
    "Hot Dog",      0,
    "French Fries", 0,
    "Root Beer",   0,
    "Grape Soda",  0,
    (char *)0,
};

/*
** Initialization of Tmenu structure for Lunch menu.
*/
Tmenu LunchMenu = { (Titem *)LunchItems, 0, 0, 0, MAP1 };

/*
** Initialize array of MyTitem's for Dinner submenu.
** Menu items do not have submenus.
*/
MyTitem DinnerItems [] = {
    "Salad",        0,
    "Steak",        0,
    "Fish",         0,
    "Baked Potato", 0,
    "Wine",         0,
    (char *)0,
};

/*
** Initialization of Tmenu structure for Dinner menu
*/
Tmenu DinnerMenu = { (Titem *)DinnerItems, 0, 0, 0, MAP1 };
```

Figure 6-4: Example Program "Tmenu2.c" (Sheet 2 of 4)

```
/*
** Initialization of array of Titem1's for root menu.
** Each Titem in the root menu has a submenu.
*/
MyTitem MainItems [] = {
    "Breakfast", &BreakfastMenu,
    "Lunch",     &LunchMenu,
    "Dinner",    &DinnerMenu,
    (char *)0,
};

/*
** Initialization of Tmenu structure for root menu
*/
Tmenu MainMenu      = { (Titem *)MainItems, 0, 0, 0, MAP1 };

main()
{
    MyTitem *item;

    request(MOUSE);
    lprintf("\n Press button 2 to get menu.");
    for(;;) {
        wait(MOUSE);
        if (button1())
            exit();
        else if (button2()) {
            /*
            ** Display Menus
            */
            item = (MyTitem *)tmenuhit(&MainMenu, BUTTON2, 0);
            if(item)
                lprintf("\n Your selection was %s",item->text);
        } else if(button3()) {
```

Figure 6-4: Example Program "Tmenu2.c" (Sheet 3 of 4)

Example Programs

```
    /*
    ** Release ownership of the mouse and
    ** sleep for two ticks of the 60Hz clock
    ** to let the control process take care of
    ** processing button 3 of the mouse.
    */
    request(0);
    sleep(2);
    /*
    ** Back on the "air," get the mouse back.
    */
    request(MOUSE);
}
}
```

Figure 6-4: Example Program "Tmenu2.c" (Sheet 4 of 4)

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

/* Library Routines and associated manual page. */
void exit();           /* EXIT(3R)           */
void lprintf();        /* PRINTF            */
int request();         /* RESOURCES(3R)     */
void sleep();          /* SLEEP(3R)         */
Titem *tmenuhit();    /* TMENUHIT(3R)      */
int wait();            /* RESOURCES(3R)     */

/* Library Macros and associated manual page. */
/*int button1();      /* BUTTONS(3R)       */
/*int button2();      /* BUTTONS(3R)       */
/*int button3();      /* BUTTONS(3R)       */

#define BUTTON2 2

/*
** definition for menumap
*/
#define MAP1      TM_TEXT | TM_NEXT

/*
** definition of customized Titem structure
*/
typedef struct MyTitem {
    char      *text;
    struct Tmenu *next;
} MyTitem;

/*
** Initialize array of MyTitem's for Breakfast submenu
** Menu items do not have submenus
*/
```

Figure 6-5: Example Program "Tmenu3.c" (Sheet 1 of 4)

Example Programs

```
*/
MyTitem BreakfastItems [] = {
    "Pancakes",    0,
    "French Toast", 0,
    "Bacon & Eggs", 0,
    "Coffee",      0,
    "Orange Juice", 0,
    (char *)0,
};

/*
** Initialization of Tmenu structure for Breakfast menu
*/
Tmenu BreakfastMenu = { (Titem *)BreakfastItems, 0, 0, 0, MAP1 };

/*
** Initialize array of MyTitem's for Lunch submenu
** Menu items do not have submenus
*/
MyTitem LunchItems [] = {
    "Hamburger",    0,
    "Hot Dog",      0,
    "French Fries", 0,
    "Root Beer",    0,
    "Grape Soda",   0,
    (char *)0,
};

/*
** Initialization of Tmenu structure for Lunch menu
*/
Tmenu LunchMenu    = { (Titem *)LunchItems, 0, 0, 0, MAP1 };
```

Figure 6-5: Example Program "Tmenu3.c" (Sheet 2 of 4)

```
/*
** Initialize array of MyTitem's for Dinner submenu
** Menu items do not have submenus
*/
MyTitem DinnerItems [] = {
    "Salad",      0,
    "Steak",      0,
    "Fish",       0,
    "Baked Potato", 0,
    "Wine",       0,
    (char *)0,
};

/*
** Initialization of Tmenu structure for Dinner menu
*/
Tmenu DinnerMenu = { (Titem *)DinnerItems, 0, 0, 0, MAP1 };

/*
** Initialization of array of Titem1's for root menu
** Each Titem in the root menu has a submenu
*/
MyTitem MainItems [] = {
    "Breakfast", &BreakfastMenu,
    "Lunch",     &LunchMenu,
    "Dinner",    &DinnerMenu,
    (char *)0,
};

/*
** Initialization of Tmenu structure for root menu
*/
Tmenu MainMenu = { (Titem *)MainItems, 0, 0, 0, MAP1 };
```

Figure 6-5: Example Program "Tmenu3.c" (Sheet 3 of 4)

Example Programs

```
main()
{
    MyTitem *item;

    request(MOUSE);
    lprintf("\n Press button 2 to get menu.");
    for(;;) {
        wait(MOUSE);
        if (button1())
            exit();
        else if (button2()) {
            /*
             ** Display menu.
             */
            item=(MyTitem *)tmenuhit(&MainMenu, BUTTON2, TM_EXPAND);
            if(item)
                lprintf("\n Your selection was %s",item->text);
        } else if(button3()) {
            /*
             ** Release ownership of the mouse and
             ** sleep for two ticks of the 60Hz clock
             ** to let the control process take care of
             ** processing button 3 of the mouse.
             */
            request(0);
            sleep(2);
            /*
             ** Back on the "air," get the mouse back.
             */
            request(MOUSE);
        }
    }
}
```

Figure 6-5: Example Program "Tmenu3.c" (Sheet 4 of 4)

```
#include <dmd.h>
#include <menu.h>
#include "MenuIcons.h"

/* Library Routines and associated manual page. */
void exit();           /* EXIT(3R)           */
void lprintf();       /* PRINTF            */
int request();        /* RESOURCES(3R)     */
void sleep();         /* SLEEP(3R)         */
Titem *tmenuhit();    /* TMENUHIT(3R)      */
int wait();           /* RESOURCES(3R)     */

/* Library Macros and associated manual page. */
/*int button1();      BUTTONS(3R)          */
/*int button2();      BUTTONS(3R)          */
/*int button3();      BUTTONS(3R)          */

#define BUTTON2 2

/*
** definition for menumap
*/
#define MAP1      TM_TEXT | TM_NEXT | TM_ICON

/*
** definition of customized Titem structure
*/
typedef struct MyTitem {
    char    *text;
    struct Tmenu    *next;
    Bitmap  *icon;
} MyTitem;

/*
** Initialize array of MyTitem's for Breakfast submenu
** Menu items do not have submenus
*/
```

Figure 6-6: Example Program "Tmenu4.c" (Sheet 1 of 5)

Example Programs

```
*/
MyTitem BreakfastItems [] = {
    "Pancakes",    0, &pancake,
    "French Toast", 0, &toast,
    "Bacon & Eggs", 0, &bacneggs,
    "Coffee",      0, &coffee,
    "Orange Juice", 0, &juice,
    (char *)0,
};

/*
** Initialization of Tmenu structure for Breakfast menu
*/
Tmenu BreakfastMenu = { (Titem *)BreakfastItems, 0, 0, 0, MAP1 };

/*
** Initialize array of MyTitem's for Lunch submenu
** Menu items do not have submenus
*/
MyTitem LunchItems [] = {
    "Hamburger",  0, &hamburger,
    "Hot Dog",    0, &hotdog,
    "French Fries", 0, &fries,
    "Root Beer",  0, &rootbeer,
    "Grape Soda", 0, &grapesoda,
    (char *)0,
};

/*
** Initialization of Tmenu structure for Lunch menu
*/
Tmenu LunchMenu      = { (Titem *)LunchItems, 0, 0, 0, MAP1 };

/*
** Initialize array of MyTitem's for Dinner submenu
```

Figure 6-6: Example Program "Tmenu4.c" (Sheet 2 of 5)

```
** Menu items do not have submenus
*/
MyTitem DinnerItems [] = {
    "Salad",      0, &salad,
    "Steak",      0, &steak,
    "Fish",       0, &fish,
    "Baked Potato", 0, &potato,
    "Wine",       0, &wineglass,
    "Anti-Acid",  0, &antiacid,
    (char *)0,
};

/*
** Initialization of Tmenu structure for Dinner menu
*/
Tmenu DinnerMenu = { (Titem *)DinnerItems, 0, 0, 0, MAP1 };

MyTitem DessertItems [] = {
    "Cupcakel",  0, &Cupcake,
    "Cake",      0, &cake,
    "Cupcake2",  0, &cup_cake,
    "Fruit",     0, &fruit,
    "Ice Cream", 0, &ice_cream,
    "Parfait",   0, &parfait,
    "Pie",       0, &pie,
    "Banana Split", 0, &split,
    (char *)0,
};

/*
** Initialization of Tmenu structure for Dinner menu
*/
Tmenu DessertMenu = { (Titem *)DessertItems, 0, 0, 0, MAP1 };

/*
** Initialization of array of Titem1's for root menu
** Each Titem in the root menu has a submenu

```

Figure 6-6: Example Program "Tmenu4.c" (Sheet 3 of 5)

Example Programs

```
*/
MyTitem MainItems [] = {
    "Breakfast", &BreakfastMenu, 0,
    "Lunch",     &LunchMenu, 0,
    "Dinner",    &DinnerMenu, 0,
    "Dessert",   &DessertMenu, 0,
    (char *)0,
};

/*
** Initialization of Tmenu structure for root menu
*/
Tmenu MainMenu      = { (Titem *)MainItems, 0, 0, 0, MAP1 };

main()
{
    MyTitem *item;

    request(MOUSE);
    lprintf("\n Press button 2 to get menu.");
    for(;;) {
        wait(MOUSE);
        if (button1())
            exit();
        else if (button2()) {
            /*
            ** Display menu.
            */
            item=(MyTitem *)tmenuhit(&MainMenu, BUTTON2, TM_EXPAND);
            if (item)
                lprintf("\n Your selection was %s",item->text);
        } else if(button3()) {
```

Figure 6-6: Example Program "Tmenu4.c" (Sheet 4 of 5)

```
    /*
    ** Release ownership of the mouse and
    ** sleep for two ticks of the 60Hz clock
    ** to let the control process take care of
    ** processing button 3 of the mouse.
    */
    request(0);
    sleep(2);
    /*
    ** Back on the "air," get the mouse back.
    */
    request(MOUSE);
}
}
```

Figure 6-6: Example Program "Tmenu4.c" (Sheet 5 of 5)

Example Programs

```
#include <dmd.h>
#include <menu.h>
#include <font.h>
#include "MenuIcons.h"

/* Library Routines and associated manual page. */
void exit();           /* EXIT(3R)           */
void lprintf();       /* PRINTF            */
int request();        /* RESOURCES(3R)     */
void sleep();         /* SLEEP(3R)         */
Titem *tmenuhit();    /* TMENUHIT(3R)      */
int wait();           /* RESOURCES(3R)     */

/* Library Macros and associated manual page. */
/*int button1();      BUTTONS(3R)         */
/*int button2();      BUTTONS(3R)         */
/*int button3();      BUTTONS(3R)         */

#define BUTTON2 2

/*
** definition for menupap
*/
#define MAP1      TM_TEXT | TM_NEXT | TM_ICON | TM_FONT

/* definition of customized Titem structure */
typedef struct MyTitem {
    char    *text;
    struct Tmenu    *next;
    Bitmap  *icon;
    Font    *font;
} MyTitem;

/*
** Initialize array of MyTitem's for Breakfast submenu
** Menu items do not have submenus
*/
```

Figure 6-7: Example Program "Tmenu5.c" (Sheet 1 of 5)

```
*/
MyTitem BreakfastItems [] = {
    "Pancakes",    0, &pancake,    0,
    "French Toast", 0, &toast,     0,
    "Bacon & Eggs", 0, &bacneggs,  0,
    "Coffee",      0, &coffee,    0,
    "Orange Juice", 0, &juice,     0,
    (char *)0,
};

/*
** Initialization of Tmenu structure for Breakfast menu
*/
Tmenu BreakfastMenu = { (Titem *)BreakfastItems, 0, 0, 0, MAP1 };

/* Initialize array of MyTitem's for Lunch submenu */
/* Menu items do not have submenus */
MyTitem LunchItems [] = {
    "Hamburger",   0, &hamburger, 0,
    "Hot Dog",     0, &hotdog,    0,
    "French Fries", 0, &fries,     0,
    "Root Beer",   0, &rootbeer,  0,
    "Grape Soda",  0, &grapesoda, 0,
    (char *)0,
};

/*
** Initialization of Tmenu structure for Lunch menu
*/
Tmenu LunchMenu    = { (Titem *)LunchItems, 0, 0, 0, MAP1 };

/*
** Initialize array of MyTitem's for Dinner submenu
** Menu items do not have submenus
```

Figure 6-7: Example Program "Tmenu5.c" (Sheet 2 of 5)

Example Programs

```
*/
MyTitem DinnerItems [] = {
    "Steak",      0, &steak,    0,
    "Fish",       0, &fish,     0,
    "Salad",      0, &salad,    0,
    "Baked Potato", 0, &potato,  0,
    "Wine",       0, &wineglass, 0,
    "Anti-Acid",  0, &antiacid, 0,
    (char *)0,
};

/*
** Initialization of Tmenu structure for Dinner menu
*/
Tmenu DinnerMenu = { (Titem *)DinnerItems, 0, 0, 0, MAP1 };

MyTitem DessertItems [] = {
    "Cupcake1",   0, &Cupcake,  0,
    "Cake",       0, &cake,      0,
    "Cupcake2",   0, &cup_cake,  0,
    "Fruit",      0, &fruit,     0,
    "Ice Cream",  0, &ice_cream, 0,
    "Parfait",    0, &parfait,   0,
    "Pie",        0, &pie,       0,
    "Banana Split", 0, &split,    0,
    (char *)0,
};

/*
** Initialization of Tmenu structure for Dinner menu
*/
Tmenu DessertMenu = { (Titem *)DessertItems, 0, 0, 0, MAP1 };

/*
** Initialization of array of Titem1's for root menu
** Each Titem in the root menu has a submenu

```

Figure 6-7: Example Program "Tmenu5.c" (Sheet 3 of 5)

```
*/
MyTitem MainItems [] = {
    "Breakfast", &BreakfastMenu, 0, 0,
    "Lunch",     &LunchMenu,     0, 0,
    "Dinner",    &DinnerMenu,    0, 0,
    "Dessert",   &DessertMenu,   0, 0,
    (char *)0,
};

/*
** Initialization of Tmenu structure for root menu
*/
Tmenu MainMenu      = { (Titem *)MainItems, 0, 0, 0, MAP1 };

main()
{
    MyTitem *item;

    /* Initialize fonts in the MyTitem arrays */
    BreakfastItems[0].font = &largefont;
    BreakfastItems[1].font = &largefont;
    BreakfastItems[2].font = &largefont;
    BreakfastItems[3].font = &smallfont;
    BreakfastItems[4].font = &smallfont;

    LunchItems[0].font = &largefont;
    LunchItems[1].font = &largefont;
    LunchItems[2].font = &mediumfont;
    LunchItems[3].font = &smallfont;
    LunchItems[4].font = &smallfont;

    DinnerItems[0].font = &largefont;
    DinnerItems[1].font = &largefont;
    DinnerItems[2].font = &mediumfont;
    DinnerItems[3].font = &mediumfont;
    DinnerItems[4].font = &smallfont;
    DinnerItems[5].font = &largefont;
}
```

Figure 6-7: Example Program "Tmenu5.c" (Sheet 4 of 5)

Example Programs

```
DessertItems[0].font = &largefont;
DessertItems[1].font = &largefont;
DessertItems[2].font = &largefont;
DessertItems[3].font = &largefont;
DessertItems[4].font = &largefont;
DessertItems[5].font = &largefont;
DessertItems[6].font = &largefont;
DessertItems[7].font = &largefont;

MainItems[0].font = &largefont;
MainItems[1].font = &largefont;
MainItems[2].font = &largefont;

request(MOUSE);
lprintf("\n Press button 2 to get menu.");
for(;;) {
    wait(MOUSE);
    if (button1())
        exit();
    else if (button2()) {
        /*
         ** Display menu.
         */
        item=(MyTitem *)tmenuhit(&MainMenu, BUTTON2, TM_EXPAND);
        if (item)
            lprintf("\n Your selection was %s",item->text);
    } else if(button3()) {
        /*
         ** Release ownership of the mouse and
         ** sleep for two ticks of the 60Hz clock
         ** to let the control process take care of
         ** processing button 3 of the mouse.
         */
        request(0);
        sleep(2);
        /*
         ** Back on the "air," get the mouse back.
         */
        request(MOUSE);
    }
}
}
```

Figure 6-7: Example Program "Tmenu5.c" (Sheet 5 of 5)

```

#include <dmd.h>
#include <menu.h>
#include <font.h>
#include "MenuIcons.h"

/* Library Routines and associated manual page. */
void exit();           /* EXIT(3R)           */
void lprintf();       /* PRINTF            */
int request();        /* RESOURCES(3R)    */
void sleep();         /* SLEEP(3R)        */
Titem *tmenuhit();    /* TMENUHIT(3R)     */
int wait();           /* RESOURCES(3R)    */

/* Library Macros and associated manual page. */
/*int button1();      BUTTONS(3R)        */
/*int button2();      BUTTONS(3R)        */
/*int button3();      BUTTONS(3R)        */

#define BUTTON2 2

/*
** definition for menumap
*/
#define MAP1  TM_TEXT|TM_UFIELD|TM_NEXT|TM_ICON|TM_FONT|TM_HFN

/*
** definition of customized Titem structure
*/
typedef struct MyTitem {
    char    *text;
    struct {
        unsigned short uval;
        unsigned short grey;
    } ufield;
    struct Tmenu    *next;
    Bitmap    *icon;
    Font    *font;
    void    (*hfn)();
} MyTitem;

```

Figure 6-8: Example Program "Tmenu6.c" (Sheet 1 of 5)

Example Programs

```
/*
** declare hfn function
*/
void SetGrey();

/*
** Initialize array of MyTitem's for Breakfast submenu
** Menu items do not have submenus
*/
MyTitem BreakfastItems [] = {
    "Pancakes",    {0, 0}, 0, &pancake,    0, SetGrey,
    "French Toast", {0, 0}, 0, &toast,    0, SetGrey,
    "Bacon & Eggs", {0, 0}, 0, &bacneggs, 0, SetGrey,
    "Coffee",      {0, 0}, 0, &coffee,   0, SetGrey,
    "Orange Juice", {0, 0}, 0, &juice,    0, SetGrey,
    (char *)0,
};

/*
** Initialization of Tmenu structure for Breakfast menu
*/
Tmenu BreakfastMenu = { (Titem *)BreakfastItems, 0, 0, 0, MAP1 };

/*
** Initialize array of MyTitem's for Lunch submenu
** Menu items do not have submenus
*/
MyTitem LunchItems [] = {
    "Hamburger",  {0, 0}, 0, &hamburger, 0, SetGrey,
    "Hot Dog",    {0, 0}, 0, &hotdog,   0, SetGrey,
    "French Fries", {0, 0}, 0, &fries,   0, SetGrey,
    "Root Beer",  {0, 0}, 0, &rootbeer, 0, SetGrey,
    "Grape Soda", {0, 0}, 0, &grapesoda, 0, SetGrey,
    (char *)0,
};
```

Figure 6-8: Example Program "Tmenu6.c" (Sheet 2 of 5)

```

/*
** Initialization of Tmenu structure for Lunch menu
*/
Tmenu LunchMenu = { (Titem *)LunchItems, 0, 0, 0, MAP1 };

/*
** Initialize array of MyTitem's for Dinner submenu
** Menu items do not have submenus
*/
MyTitem DinnerItems [] = {
    "Steak",      {0, 0}, 0, &steak,    0, SetGrey,
    "Fish",       {0, 0}, 0, &fish,     0, SetGrey,
    "Salad",      {0, 0}, 0, &salad,    0, SetGrey,
    "Baked Potato", {0, 0}, 0, &potato,   0, SetGrey,
    "Wine",       {0, 0}, 0, &wineglass, 0, SetGrey,
    "Anti-Acid",  {0, 0}, 0, &antiacid,  0, SetGrey,
    (char *)0,
};

/*
** Initialization of Tmenu structure for Dinner menu
*/
Tmenu DinnerMenu = { (Titem *)DinnerItems, 0, 0, 0, MAP1 };

/*
** Initialize array of MyTitem's for Dessert submenu
** Menu items do not have submenus
*/
MyTitem DessertItems [] = {
    "Cupcake1",   {0, 0}, 0, &Cupcake,    0, SetGrey,
    "Cake",       {0, 0}, 0, &cake,      0, SetGrey,
    "Cupcake2",   {0, 0}, 0, &cup_cake,   0, SetGrey,
    "Fruit",      {0, 0}, 0, &fruit,     0, SetGrey,
    "Ice Cream",  {0, 0}, 0, &ice_cream,  0, SetGrey,
    "Parfait",    {0, 0}, 0, &parfait,   0, SetGrey,
    "Pie",        {0, 0}, 0, &pie,       0, SetGrey,
    "Banana Split", {0, 0}, 0, &split,    0, SetGrey,
    (char *)0,
};

```

Figure 6-8: Example Program "Tmenu6.c" (Sheet 3 of 5)

Example Programs

```
/*
** Initialization of Tmenu structure for Dessert menu
*/
Tmenu DessertMenu = { (Titem *)DessertItems, 0, 0, 0, MAP1 };

/*
** Initialization of array of Titem1's for root menu
** Each Titem in the root menu has a submenu
*/
MyTitem MainItems [] = {
    "Breakfast", {0, 0}, &BreakfastMenu, 0, 0, 0,
    "Lunch",      {0, 0}, &LunchMenu,    0, 0, 0,
    "Dinner",    {0, 0}, &DinnerMenu,   0, 0, 0,
    "Dessert",   {0, 0}, &DessertMenu,  0, 0, 0,
    (char *)0,
};

/*
** Initialization of Tmenu structure for root menu
*/
Tmenu MainMenu = { (Titem *)MainItems, 0, 0, 0, MAP1 };

main()
{
    MyTitem *item;

    /* Initialize fonts in the MyTitem arrays */
    BreakfastItems[0].font = &largefont;
    BreakfastItems[1].font = &largefont;
    BreakfastItems[2].font = &largefont;
    BreakfastItems[3].font = &smallfont;
    BreakfastItems[4].font = &smallfont;

    LunchItems[0].font = &largefont;
    LunchItems[1].font = &largefont;
    LunchItems[2].font = &mediumfont;
    LunchItems[3].font = &smallfont;
    LunchItems[4].font = &smallfont;
}
```

Figure 6-8: Example Program "Tmenu6.c" (Sheet 4 of 5)

```
DinnerItems[0].font = &largefont;
DinnerItems[1].font = &largefont;
DinnerItems[2].font = &mediumfont;
DinnerItems[3].font = &mediumfont;
DinnerItems[4].font = &smallfont;
DinnerItems[5].font = &largefont;

DessertItems[0].font = &largefont;
DessertItems[1].font = &largefont;
DessertItems[2].font = &largefont;
DessertItems[3].font = &largefont;
DessertItems[4].font = &largefont;
DessertItems[5].font = &largefont;
DessertItems[6].font = &largefont;
DessertItems[7].font = &largefont;

MainItems[0].font = &largefont;
MainItems[1].font = &largefont;
MainItems[2].font = &largefont;

request(MOUSE);
lprintf("\n Press button 2 to get menu.");
for(;;) {
    wait(MOUSE);
    if (button1())
        exit();
    else if (button2()) {
        /*
         ** Display menu.
         */
        item=(MyTitem *)tmenuhit(&MainMenu, BUTTON2, TM_EXPAND);
        if (item)
            lprintf("\n Your selection was %s",item->text);
    }
}

void
SetGrey(item)
MyTitem *item;
{
    item->ufield.grey = 1;
}
```

Figure 6-8: Example Program "Tmenu6.c" (Sheet 5 of 5)

Example Programs

```
#include <dmd.h>
#include <menu.h>
#include <font.h>
#include "MenuIcons.h"

/* Library Routines and associated manual page. */
void exit();           /* EXIT(3R)           */
void lprintf();       /* PRINTF            */
int request();        /* RESOURCES(3R)    */
void sleep();         /* SLEEP(3R)        */
Titem *tmenuhit();   /* TMENUHIT(3R)     */
int wait();           /* RESOURCES(3R)    */

/* Library Macros and associated manual page. */
/*int button1();      BUTTONS(3R)        */
/*int button2();      BUTTONS(3R)        */
/*int button3();      BUTTONS(3R)        */

#define BUTTON2 2

/*
** definition for menumap
*/
#define MAP1  TM_TEXT|TM_UFIELD|TM_NEXT|TM_ICON|TM_FONT|TM_HFN

/*
** definition for flags parameter passed to tmenuhit
*/
#define FLAGS  TM_STATIC

/*
** definition of customized Titem structure
*/
typedef struct MyTitem {
    char    *text;
    struct {
        unsigned short uval;
        unsigned short grey;
    } ufield;
    struct Tmenu    *next;
    Bitmap    *icon;
};
```

Figure 6-9: Example Program "Tmenu7.c" (Sheet 1 of 6)

```

    Font    *font;
    void    (*hfn)();
} MyTitem;

/*
** declare hfn function
*/
void SetGrey();

/*
** Initialize array of MyTitem's for Breakfast submenu
** Menu items do not have submenus
*/
MyTitem BreakfastItems [] = {
    "Pancakes",    {0, 0}, 0, &pancake,    0, SetGrey,
    "French Toast", {0, 0}, 0, &toast,    0, SetGrey,
    "Bacon & Eggs", {0, 0}, 0, &bacneggs,  0, SetGrey,
    "Coffee",     {0, 0}, 0, &coffee,   0, SetGrey,
    "Orange Juice", {0, 0}, 0, &juice,    0, SetGrey,
    (char *)0,
};

/*
** Initialization of Tmenu structure for Breakfast menu
*/
Tmenu BreakfastMenu = { (Titem *)BreakfastItems, 0, 0, 0, MAP1 };

/*
** Initialize array of MyTitem's for Lunch submenu
** Menu items do not have submenus
*/
MyTitem LunchItems [] = {
    "Hamburger",  {0, 0}, 0, &hamburger, 0, SetGrey,
    "Hot Dog",    {0, 0}, 0, &hotdog,   0, SetGrey,
    "French Fries", {0, 0}, 0, &fries,   0, SetGrey,
    "Root Beer",  {0, 0}, 0, &rootbeer, 0, SetGrey,
    "Grape Soda", {0, 0}, 0, &grapesoda, 0, SetGrey,
    (char *)0,
};

```

Figure 6-9: Example Program "Tmenu7.c" (Sheet 2 of 6)

Example Programs

```
/*
** Initialization of Tmenu structure for Lunch menu
*/
Tmenu LunchMenu = { (Titem *)LunchItems, 0, 0, 0, MAP1 };

/*
** Initialize array of MyTitem's for Dinner submenu
** Menu items do not have submenus
*/
MyTitem DinnerItems [] = {
    "Steak",      {0, 0}, 0, &steak,    0, SetGrey,
    "Fish",       {0, 0}, 0, &fish,     0, SetGrey,
    "Salad",      {0, 0}, 0, &salad,    0, SetGrey,
    "Baked Potato", {0, 0}, 0, &potato,   0, SetGrey,
    "Wine",       {0, 0}, 0, &wineglass, 0, SetGrey,
    "Anti-Acid",  {0, 0}, 0, &antiacid,  0, SetGrey,
    (char *)0,
};

/*
** Initialization of Tmenu structure for Dinner menu
*/
Tmenu DinnerMenu = { (Titem *)DinnerItems, 0, 0, 0, MAP1 };

/*
** Initialize array of MyTitem's for Dessert submenu
** Menu items do not have submenus
*/
MyTitem DessertItems [] = {
    "Cupcake1",   {0, 0}, 0, &Cupcake,    0, SetGrey,
    "Cake",       {0, 0}, 0, &cake,     0, SetGrey,
    "Cupcake2",   {0, 0}, 0, &cup_cake,   0, SetGrey,
    "Fruit",      {0, 0}, 0, &fruit,     0, SetGrey,
    "Ice Cream",  {0, 0}, 0, &ice_cream,  0, SetGrey,
    "Parfait",    {0, 0}, 0, &parfait,   0, SetGrey,
    "Pie",        {0, 0}, 0, &pie,      0, SetGrey,
    "Banana Split", {0, 0}, 0, &split,    0, SetGrey,
    (char *)0,
};
```

Figure 6-9: Example Program "Tmenu7.c" (Sheet 3 of 6)

```
/*
** Initialization of Tmenu structure for Dessert menu
*/
Tmenu DessertMenu = { (Titem *)DessertItems, 0, 0, 0, MAP1 };

/*
** Initialization of array of Titem1's for root menu
** Each Titem in the root menu has a submenu
*/
MyTitem MainItems [] = {
    "Breakfast", {0, 0}, &BreakfastMenu, 0, 0, 0,
    "Lunch",     {0, 0}, &LunchMenu,     0, 0, 0,
    "Dinner",    {0, 0}, &DinnerMenu,    0, 0, 0,
    "Dessert",   {0, 0}, &DessertMenu,   0, 0, 0,
    (char *)0,
};

/*
** Initialization of Tmenu structure for root menu
*/
Tmenu MainMenu = { (Titem *)MainItems, 0, 0, 0, MAP1 };

main()
{
    MyTitem *item;

    /* Initialize fonts in the MyTitem arrays */
    BreakfastItems[0].font = &largefont;
    BreakfastItems[1].font = &largefont;
    BreakfastItems[2].font = &largefont;
    BreakfastItems[3].font = &smallfont;
    BreakfastItems[4].font = &smallfont;

    LunchItems[0].font = &largefont;
    LunchItems[1].font = &largefont;
    LunchItems[2].font = &mediumfont;
    LunchItems[3].font = &smallfont;
    LunchItems[4].font = &smallfont;
}
```

Figure 6-9: Example Program "Tmenu7.c" (Sheet 4 of 6)

Example Programs

```
DinnerItems[0].font = &largefont;
DinnerItems[1].font = &largefont;
DinnerItems[2].font = &mediumfont;
DinnerItems[3].font = &mediumfont;
DinnerItems[4].font = &smallfont;
DinnerItems[5].font = &largefont;

DessertItems[0].font = &largefont;
DessertItems[1].font = &largefont;
DessertItems[2].font = &largefont;
DessertItems[3].font = &largefont;
DessertItems[4].font = &largefont;
DessertItems[5].font = &largefont;
DessertItems[6].font = &largefont;
DessertItems[7].font = &largefont;

MainItems[0].font = &largefont;
MainItems[1].font = &largefont;
MainItems[2].font = &largefont;

request(MOUSE);
lprintf("\n Click button 2 to get Menu.\n");
for(;;) {
    wait(MOUSE);
    if (button1())
        exit();
    else if (button2()) {
        /*
         ** Wait for all buttons to be released.
         */
        bttns(0);

        /*
         ** Display Menu.
         */
        (void)tmenuhit(&MainMenu, BUTTON2, FLAGS);

        /*
         ** Wait for all buttons to be released.
         */
        bttns(0);
    }
}
```

Figure 6-9: Example Program "Tmenu7.c" (Sheet 5 of 6)

```
    } else if(button3()) {
        /*
        ** Release ownership of the mouse and
        ** sleep for two ticks of the 60Hz clock
        ** to let the control process take care of
        ** processing button 3 of the mouse.
        */
        request(0);
        sleep(2);
        /*
        ** Back on the "air," get the mouse back.
        */
        request(MOUSE);
    }
}

void
SetGrey(item)
MyTitem *item;
{
    item->ufield.grey = 1;
}
```

Figure 6-9: Example Program "Tmenu7.c" (Sheet 6 of 6)

Example Programs

```
#include <dmd.h>
#include <menu.h>
#include <font.h>
#include "MenuIcons.h"

/* Library Routines and associated manual page. */
void exit();           /* EXIT(3R)           */
void lprintf();       /* PRINTF           */
int request();        /* RESOURCES(3R)    */
void sleep();         /* SLEEP(3R)        */
Titem *tmenuhit();    /* TMENUHIT(3R)     */
Point sPtCurrent();   /* MOVETO(3L)       */
int wait();           /* RESOURCES(3R)    */

/* Library Macros and associated manual page. */
/*int button1();      BUTTONS(3R)        */
/*int button2();      BUTTONS(3R)        */
/*int button3();      BUTTONS(3R)        */

#define BUTTON2 2

/*
** definition for menumap
*/
#define MAP1  TM_TEXT|TM_UFIELD|TM_NEXT|TM_ICON|TM_FONT|TM_HFN

/*
** definition for flags parameter passed to tmenuhit
*/
#define FLAGS  TM_STATIC | TM_NORET | TM_POINT

/*
** definition of customized Titem structure
*/
typedef struct MyTitem {
    char    *text;
    struct {
        unsigned short uval;
        unsigned short grey;
    } ufield;
    struct Tmenu    *next;
    Bitmap    *icon;
    Font    *font;
};
```

Figure 6-10: Example Program "Tmenu8.c" (Sheet 1 of 6)

```

    void    (*hfn)();
} MyTitem;

/*
** declare hfn function
*/
void SetGrey();

/*
** Initialize array of MyTitem's for Breakfast submenu
** Menu items do not have submenus
*/
MyTitem BreakfastItems [] = {
    "Pancakes",    {0, 0}, 0, &pancake,    0, SetGrey,
    "French Toast", {0, 0}, 0, &toast,      0, SetGrey,
    "Bacon & Eggs", {0, 0}, 0, &bacneggs,   0, SetGrey,
    "Coffee",      {0, 0}, 0, &coffee,    0, SetGrey,
    "Orange Juice", {0, 0}, 0, &juice,      0, SetGrey,
    (char *)0,
};

/*
** Initialization of Tmenu structure for Breakfast menu
*/
Tmenu BreakfastMenu = { (Titem *)BreakfastItems, 0, 0, 0, MAP1 };

/*
** Initialize array of MyTitem's for Lunch submenu
** Menu items do not have submenus
*/
MyTitem LunchItems [] = {
    "Hamburger",   {0, 0}, 0, &hamburger, 0, SetGrey,
    "Hot Dog",     {0, 0}, 0, &hotdog,   0, SetGrey,
    "French Fries", {0, 0}, 0, &fries,    0, SetGrey,
    "Root Beer",   {0, 0}, 0, &rootbeer, 0, SetGrey,
    "Grape Soda",  {0, 0}, 0, &grapesoda, 0, SetGrey,
    (char *)0,
};

```

Figure 6-10: Example Program "Tmenu8.c" (Sheet 2 of 6)

Example Programs

```
/*
** Initialization of Tmenu structure for Lunch menu
*/
Tmenu LunchMenu      = { (Titem *)LunchItems, 0, 0, 0, MAP1 };

/*
** Initialize array of MyTitem's for Dinner submenu
** Menu items do not have submenus
*/
MyTitem DinnerItems [] = {
    "Steak",      {0, 0}, 0, &steak,    0, SetGrey,
    "Fish",       {0, 0}, 0, &fish,     0, SetGrey,
    "Salad",      {0, 0}, 0, &salad,    0, SetGrey,
    "Baked Potato", {0, 0}, 0, &potato,   0, SetGrey,
    "Wine",       {0, 0}, 0, &wineglass, 0, SetGrey,
    "Anti-Acid",  {0, 0}, 0, &antiacid,  0, SetGrey,
    (char *)0,
};

/*
** Initialization of Tmenu structure for Dinner menu
*/
Tmenu DinnerMenu     = { (Titem *)DinnerItems, 0, 0, 0, MAP1 };

/*
** Initialize array of MyTitem's for Dessert submenu
** Menu items do not have submenus
*/
MyTitem DessertItems [] = {
    "Cupcake1",  {0, 0}, 0, &Cupcake,    0, SetGrey,
    "Cake",      {0, 0}, 0, &cake,      0, SetGrey,
    "Cupcake2",  {0, 0}, 0, &cup_cake,   0, SetGrey,
    "Fruit",     {0, 0}, 0, &fruit,     0, SetGrey,
    "Ice Cream", {0, 0}, 0, &ice_cream,  0, SetGrey,
    "Parfait",   {0, 0}, 0, &parfait,   0, SetGrey,
    "Pie",       {0, 0}, 0, &pie,       0, SetGrey,
    "Banana Split", {0, 0}, 0, &split,    0, SetGrey,
    (char *)0,
};
```

Figure 6-10: Example Program "Tmenu8.c" (Sheet 3 of 6)

```
/*
** Initialization of Tmenu structure for Dessert menu
*/
Tmenu DessertMenu = { (Titem *)DessertItems, 0, 0, 0, MAP1 };

/*
** Initialization of array of Titem1's for root menu
** Each Titem in the root menu has a submenu
*/
MyTitem MainItems [] = {
    "Breakfast", {0, 0}, &BreakfastMenu, 0, 0, 0,
    "Lunch",      {0, 0}, &LunchMenu,    0, 0, 0,
    "Dinner",    {0, 0}, &DinnerMenu,   0, 0, 0,
    "Dessert",   {0, 0}, &DessertMenu,  0, 0, 0,
    (char *)0,
};

/*
** Initialization of Tmenu structure for root menu
*/
Tmenu MainMenu = { (Titem *)MainItems, 0, 0, 0, MAP1 };

main()
{
    MyTitem *item;

    /* Initialize fonts in the MyTitem arrays */
    BreakfastItems[0].font = &largefont;
    BreakfastItems[1].font = &largefont;
    BreakfastItems[2].font = &largefont;
    BreakfastItems[3].font = &smallfont;
    BreakfastItems[4].font = &smallfont;

    LunchItems[0].font = &largefont;
    LunchItems[1].font = &largefont;
    LunchItems[2].font = &mediumfont;
    LunchItems[3].font = &smallfont;
    LunchItems[4].font = &smallfont;
}
```

Figure 6-10: Example Program "Tmenu8.c" (Sheet 4 of 6)

Example Programs

```
DinnerItems[0].font = &largefont;
DinnerItems[1].font = &largefont;
DinnerItems[2].font = &mediumfont;
DinnerItems[3].font = &mediumfont;
DinnerItems[4].font = &smallfont;
DinnerItems[5].font = &largefont;

DessertItems[0].font = &largefont;
DessertItems[1].font = &largefont;
DessertItems[2].font = &largefont;
DessertItems[3].font = &largefont;
DessertItems[4].font = &largefont;
DessertItems[5].font = &largefont;
DessertItems[6].font = &largefont;
DessertItems[7].font = &largefont;

MainItems[0].font = &largefont;
MainItems[1].font = &largefont;
MainItems[2].font = &largefont;

request(MOUSE);
lprintf("\n Click button 2 to get Menu.\n");
for(;;) {
    wait(MOUSE);
    if (button1())
        exit();
    else if (button2()) {
        /*
         ** Wait for all buttons to be released.
         */
        bttns(0);

        /*
         ** Display Menu.
         */
        (void)tmenuhit(&MainMenu, BUTTON2, FLAGS, sPtCurrent());

        /*
         ** Wait for all buttons to be released.
         */
        bttns(0);
    }
}
```

Figure 6-10: Example Program "Tmenu8.c" (Sheet 5 of 6)

```
    } else if(button3()) {
        /*
         ** Release ownership of the mouse and
         ** sleep for two ticks of the 60Hz clock
         ** to let the control process take care of
         ** processing button 3 of the mouse.
         */
        request(0);
        sleep(2);
        /*
         ** Back on the "air," get the mouse back.
         */
        request(MOUSE);
    }
}

void
SetGrey(item)
MyTitem *item;
{
    item->ufield.grey = 1;
}
```

Figure 6-10: Example Program "Tmenu8.c" (Sheet 6 of 6)

Example Programs

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

/* Library Routines and associated manual page. */
Point add();          /* PTARITH(3R)      */
void exit();         /* EXIT(3R)       */
void rectf();        /* RECTF(3R)      */
int request();       /* RESOURCES(3R)  */
void sleep();        /* SLEEP(3R)      */
char *strcpy();      /* STRING(3L)     */
Point string();      /* STRING(3R)     */
Titem *tmenuhit();   /* TMENUHIT(3R)   */
int wait();          /* RESOURCES(3R)  */

/* Library Macros and associated manual page. */
/*int button1();     BUTTONS(3R)      */
/*int button2();     BUTTONS(3R)      */
/*int button3();     BUTTONS(3R)      */
/*Point Pt();        PT(3L)           */

void dfn(), bfn(), hfn();
extern Tmenu dmenu;

Titem ms[] =
{
    "0", 0, 0, &dmenu, 0, 0, dfn, bfn, hfn,
    "1", 0, 0, &dmenu, 0, 0, dfn, bfn, hfn,
    "2", 0, 0, &dmenu, 0, 0, dfn, bfn, hfn,
    "3", 0, 0, &dmenu, 0, 0, dfn, bfn, hfn,
    "4", 0, 0, &dmenu, 0, 0, dfn, bfn, hfn,
    "5", 0, 0, &dmenu, 0, 0, dfn, bfn, hfn,
    "6", 0, 0, &dmenu, 0, 0, dfn, bfn, hfn,
    "7", 0, 0, &dmenu, 0, 0, dfn, bfn, hfn,
    "8", 0, 0, &dmenu, 0, 0, dfn, bfn, hfn,
    "9", 0, 0, &dmenu, 0, 0, dfn, bfn, hfn,
    0
};
Tmenu dmenu = { ms };
char digs[64], result[64];
int store;
int ndig = 0;
```

Figure 6-11: Example Program "Tmenu9.c" (Sheet 1 of 3)

```

main()
{
    Point p;

    request(MOUSE);
    while (wait(MOUSE))
        if (button1())
            exit();
        else if (button2()) {
            store = 1;
            if (tmenuhit(&dmenu, 2, 0)) {
                rectf(&display, Direct, F_CLR);
                p = add(Direct.origin, Pt(8,8));
                string(&largefont, result, &display,
                    p, F_XOR);
            }
        } else if (button3()) {
            /*
            ** Release ownership of the mouse and
            ** sleep for two ticks of the 60Hz clock
            ** to let the control process take care of
            ** processing button 3 of the mouse.
            */
            request(0);
            sleep(2);
            /*
            ** Back on the "air," get the mouse back.
            */
            request(MOUSE);
        }
    }

void
dfn(mi)
    register Titem *mi;
{
    store = 1;
    digs[ndig++] = mi->text[0];
    digs[ndig] = 0;
}

```

Figure 6-11: Example Program "Tmenu9.c" (Sheet 2 of 3)

Example Programs

```
void
bfn(mi)
    register Titem *mi;
{
    digs[--ndig] = 0;
}

void
hfn(mi)
    register Titem *mi;
{
    if (store)
    {
        dfn(mi);
        strcpy(result, digs);
        bfn(mi);
        store = 0;
    }
}
```

Figure 6-11: Example Program "Tmenu9.c" (Sheet 3 of 3)

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

/* Library Routines and associated manual page. */
void labelicon();      /* LABELON(3R)      */
void labelon();        /* LABELON(3R)      */
int request();         /* RESOURCES(3R)    */
int wait();            /* RESOURCES(3R)    */

Bitmap B_skull;

Rectangle fRect();

main()
{
    /*
    ** Dynamically initialize the Bitmap B_skull
    ** with the data from the texture16 cursor
    ** C_skull.
    */
    B_skull.base = (Word *)&C_skull;
    B_skull.width = 1;
    B_skull.rect = fRect(0, 0, 16, 16);

    request(MOUSE);

    /*
    ** Turn on the label bar.
    */
    labelon();

    /*
    ** Display the skull icon twice.
    */
    labelicon(&B_skull, L_USER_POSITION);
    labelicon(&B_skull, L_USER_POSITION + 1);

    /*
    ** Exit when the user hits button 1.
    */
    while (!button1()) wait(MOUSE);
}
```

Figure 6-12: Example Program "label1.c"

Example Programs

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

/* Library Routines and associated manual page. */
void labelicon();      /* LABELON(3R)      */
void labelon();        /* LABELON(3R)      */
void labeltext();      /* LABELON(3R)      */
int request();         /* RESOURCES(3R)    */
int wait();            /* RESOURCES(3R)    */

Bitmap B_skull;
Rectangle fRect();

main()
{
    /*
    ** Dynamically initialize the Bitmap B_skull
    ** with the data from the texture16 cursor
    ** C_skull.
    */
    B_skull.base = (Word *)&C_skull;
    B_skull.width = 1;
    B_skull.rect = fRect(0, 0, 16, 16);

    request(MOUSE);

    /*
    ** Turn on the label bar.
    */
    labelon();

    /*
    ** Display three text strings.
    */
    labeltext("left", 4, L_LEFT);
    labeltext("center", 6, L_CENTER);
    labeltext("right", 5, L_RIGHT);

    /*
    ** Exit when the user hits button 1.
    */
    while (!button1()) wait(MOUSE);
}
```

Figure 6-13: Example Program "label2.c"

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

/* Library Routines and associated manual page. */
void labelicon();      /* LABELON(3R)      */
void labelon();       /* LABELON(3R)      */
void labeltext();     /* LABELON(3R)      */
int request();        /* RESOURCES(3R)    */
int wait();           /* RESOURCES(3R)    */

Bitmap B_skull;

Rectangle fRect();

main()
{
    /*
    ** Dynamically initialize the Bitmap B_skull
    ** with the data from the texture16 cursor
    ** C_skull.
    */
    B_skull.base = (Word *)&C_skull;
    B_skull.width = 1;
    B_skull.rect = fRect(0, 0, 16, 16);

    request(MOUSE);

    /*
    ** Turn on the label bar.
    */
    labelon();

    /*
    ** Display the skull icon twice.
    */
    labelicon(&B_skull, L_USER_POSITION);
    labelicon(&B_skull, L_USER_POSITION + 1);
}
```

Figure 6-14: Example Program "label3.c" (Sheet 1 of 2)

Example Programs

```
/*
** Display three text strings.
*/
labeltext("  left", 8, L_LEFT);
labeltext("center", 6, L_CENTER);
labeltext("right", 5, L_RIGHT);

/*
** Exit when the user hits button 1.
*/
while (!button1()) wait(MOUSE);
}
```

Figure 6-14: Example Program "label3.c" (Sheet 2 of 2)

```
#include <dmd.h>

/* Library routines and associated manual page. */
void btns();          /* BUTTONS(3R) */
void exit();          /* EXIT(3R) */
int kbdchar();        /* KBDCHAR(3R) */
int msgbox();         /* MSGBOX(3R) */
int request();        /* RESOURCES(3R) */
int wait();           /* RESOURCES(3R) */

/* Library macros and associated manual page. */
/* int button123()      BUTTONS(3R) */

main()
{
    request(MOUSE|KBD);

    for(;;) {
        wait(MOUSE);
        if(button123()) {
            /*
             ** Draw message box in response
             ** to any button pressed.
             */
            msgbox("This is a Message",
                   "Box Demonstration",
                   (char *)0);
            /*
             ** Busy loop until all mouse
             ** mouse buttons are released.
             */
            btns(0);
        }
        /*
         ** Exit if the user has typed any keys.
         */
        if(kbdchar() != -1)
            exit();
    }
}
```

Figure 6-15: Example Program "msgbox1.c"

Example Programs

```
#include <dmd.h>

/* Library routines and associated manual page. */
void bttns();           /* BUTTONS(3R)           */
void exit();           /* EXIT(3R)             */
int kbdchar();         /* KBDCHAR(3R)         */
int menuhit();         /* MENUHIT(3L)         */
int msgbox();         /* MSGBOX(3R)          */
int request();         /* RESOURCES(3R)       */
void ringbell();      /* RINGBELL(3R)        */
int wait();           /* RESOURCES(3R)       */

/* Library macros and associated manual page. */
/* int button1()          BUTTONS(3R)          */
/* int button2()          BUTTONS(3R)          */

char *menutext[] = {
    "Message Boxes",
    "Help",
    "Default",
    "Small Box",
    "Medium Box",
    "Big Box",
    "DON'T CHOOSE!",
    "Style",
    "Exit",
    (char *)0
};

Menu menu = { menutext };

char *stylemenutext[] = {
    "Confirm",
    (char *)0
};

Menu stylemenu = { stylemenutext };

main()
{
    request(MOUSE);
    for(;;) {
```

Figure 6-16: Example Program "msgbox2.c" (Sheet 1 of 3)

```
wait(MOUSE);
if(button1())
    break;
if(button2()) {
    switch( menuhit(&menu, 2) ) {
        case 0:
            msgbox("Hey!",
                "Hey!",
                "MESSAGE BOXES",
                (char *)0
            );
            bttns(0);
            break;
        case 1:
            msgbox("Help is on the way",
                (char *)0
            );
            bttns(0);
            break;
        case 2: /* default - no memory */
            msgbox((char *)0);
            bttns(0);
            break;
        case 3:
            msgbox("X", (char *)0);
            bttns(0);
            break;
        case 4:
            msgbox("This is an example",
                "of a slightly",
                "larger message box.",
                "The quick brown fox jumps over the lazy dog",
                (char *)0
            );
            bttns(0);
            break;
        case 5:
            msgbox("This is a",
                "REALLY",
                "BIG",
                "message",
```

Figure 6-16: Example Program "msgbox2.c" (Sheet 2 of 3)

Example Programs

```
        "box",
        "The quick brown fox jumps over the lazy
        dog three times, turns around twice and
        then drops dead",
        (char *)0
    );
    btns(0);
    break;
case 6:
    msgbox( "PLEASE",
        "do",
        "not",
        "choose",
        "this",
        "item",
        "again",
        (char *)0
    );
    btns(0);
    break;
case 7:
    msgbox( "A different style",
        "-----",
        "Do you want to ring the bell?",
        "Press button 2 to confirm",
        (char *)0
    );
    if(button2())
        if(menuhit(&stylemenu, 2) == 0)
            ringbell();
    btns(0);
    break;
case 8:
    msgbox( "Exit the message box demo",
        "by clicking button 1",
        "When there is no box on the screen",
        (char *)0
    );
    btns(0);
    break;
}
}
}
```

Figure 6-16: Example Program "msgbox2.c" (Sheet 3 of 3)

Chapter 7: “jx” I/O Interpreter

Introduction	7-1
How “jx” Works	7-2
“stdout” and “stderr”	7-2
“stdin”	7-2
Using “jx”	7-3
Functions Available with “jx”	7-4
Example Program	7-5

Introduction

jsx is the 630 MTG's standard I/O interpreter. The **jsx** utility calls **dmdld** to download given applications to the 630 MTG. Once downloaded, **jsx** continues to run to interpret I/O calls. The downloaded application can then execute I/O routines with the host and with files on the host using standard UNIX System V interface programs such as **fopen** and **printf**.

How “jx” Works

jx is an example of a program which "offloads" the host computer by dividing the work between the host processor and the terminal. This "offloading" involves the simultaneous operation of two programs: **jx** in the 630 MTG and **sysint** on the host. The **jx** utility first forks a **dmdld** process and waits for it to complete.

Note: Many of the **dmdld** options are available on the **jx** command line. See the **JX(1)** and **DMDLD(1)** manual pages for more information.

When the download completes, **sysint** starts running on the host. **sysint** "listens" for the application to send I/O requests to the host and interprets the message.

“stdout” and “stderr”

While the application is running, **stdout** and **stderr** are redirected to two files in \$HOME called **.jxout** and **.jxerr**, respectively. After the application completes and the default terminal emulator is started again in the window, **stdout** and **stderr** are directed back to the terminal and the contents of **.jxout** and **.jxerr** are displayed in the window.

“stdin”

stdin is also properly redirected to the host by using the **jx** command line and the **popen** function. The host does NOT receive input directly from the 630 MTG's keyboard. Programs that want to read from the keyboard need to use **KBDCHAR(3R)**.

Using “jx”

There are some other important features about **jx** to keep in mind when writing or modifying an application in order to properly use **jx**. These features are pointed out in the following procedure for using **jx**. The example program **jxmouse.c**, included at the end of this chapter, also exhibits these features. The source code for **jxmouse.c** can be found in *\$DMD/examples/jx*.

1. Include the file **dmdio.h**. This file resembles the standard include file **stdio.h** but contains some specific items for the 630 MTG. Note that **stdio.h** must not be included.
2. Have your application call **exit** upon completion. See **exit(3R)** in the *630 MTG Software Reference Manual*. When an application is not running under **jx**, **exit** is a routine that causes your downloaded application to terminate and be replaced by **Windowproc**. With **jx**, **exit** also takes care of cleaning up the UNIX System files accessed by your application. If an application downloaded with **jx** does not call **exit** when it ends, the window may look "dead" since the exit is incomplete. The situation is not hopeless—typing the **DELETE** key in that window should trigger a complete exit.
3. Compile the program as usual using **dmdcc**.
4. Use **jx** to download your application instead of **dmdld**. An application using the standard I/O routines will not work without **jx** to interface with the host. Give the command:

```
jx <file>
```

Note: Command line arguments and **dmdld** options are all allowed.

Functions Available with “jx”

The standard UNIX System V I/O routines listed below can be used with **jx**. In addition, the **printf** type routines that send output to the screen (such as **lprintf** and **bprintf**) can still be used.

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

Note: Programs which run under **jx** should use the above routines, rather than **sendchar** and **rcvchar**, to send and receive characters from the host.

Example Program

The following program starts with the mouse tracking example from the discussion on the mouse resource. As long as its window is current, the application will track mouse movement with the "world" icon and send the mouse coordinates to **stdout** using **printf**. When button three is clicked, mouse tracking will stop and the application will read keyboard input until **@** is typed.

Notice the special lines in the example source code for running under **jx**: the include of the file **dmdio.h**, and the **exit** call at the end. Remember, **printf** writes to **stdout**, which is redirected to **\$HOME/.jxout** automatically by **jx**. The **fprintf** to **stderr** will write to **\$HOME/.jxerr**, and the contents of both of these files will be dumped to the window after the program exits. The results of **lprintf** are displayed in the window while the application is running. The program also demonstrates how to use **kbdchar** to read input from the keyboard.

A good way to observe what is taking place in this program is to open a second window and execute:

```
tail -f $HOME/.jxout
```

This command line displays the x- and y-coordinates of the mouse as you move it around.

Figure 7-1 gives a printout of the source code for the **jxmouse** example program.

Example Program

```
#include <dmd.h>
#include <dmdio.h>          /* Needed for jx */
#include "world.h"

/* Library Routines and associated manual page. */
void bitblt();           /* BITBLT(3R) */
int eqpt();              /* EQ(3R) */
void exit();             /* EXIT(3R) */
void fprintf();          /* PRINTF(3L) */
int kbdchar();           /* KBDCHAR(3R) */
void lprintf();          /* PRINTF(3L) */
int own();                /* RESOURCES(3R) */
int request();           /* RESOURCES(3R) */
void sleep();            /* SLEEP(3R) */
int wait();              /* RESOURCES(3R) */

/* Library Macros and associated manual page. */
/* int button3           BUTTONS(3R) */

#define MAXLEN 250

main()
{
    Point MousePosition;
    Point OldPosition;
    unsigned char c;
    char *p;
    char words[MAXLEN];
    short len;

    len = 0;
    p = words;
```

Figure 7-1: Example Program - "jxmouse" (Sheet 1 of 4)

```
/*
** Request the use of the MOUSE resource.
*/
request(MOUSE);

/*
** Allow the 630 MTG control process to run
** and update the mouse position.
*/
sleep(2);

/*
** Record the current mouse position.
*/
MousePosition = mouse.xy;

/*
** Draw the world Bitmap at the current
** mouse position.
*/
bitblt(&world, world.rect, &display, MousePosition, F_XOR);

/*
** Track the mouse until button 3 pressed
*/
for(wait(MOUSE); !button3();wait(MOUSE)) {
    /*
    ** Erase the world Bitmap from the old
    ** mouse position.
    */
    bitblt(&world, world.rect, &display, MousePosition, F_XOR);

    /*
    ** Update the MousePosition.
    */
    OldPosition = MousePosition;
    MousePosition = mouse.xy;
}
```

Figure 7-1: Example Program - "jxmouse" (Sheet 2 of 4)

Example Program

```
/*
** Draw the world at the new position.
*/
bitblt(&world, world.rect, &display, MousePosition, F_XOR);
/*
** Sleep for two ticks of the 60Hz clock to
** release the CPU and synchronize with the
** 60Hz refresh rate of the 630 MTG screen.
*/
sleep(2);

/*
** If current window, write the mouse
** coordinates to stdout
*/
if ((own() & MOUSE) && !eqpt(OldPosition, MousePosition)) {
    printf("x = %d\n", mouse.xy.x);
    printf("y = %d\n\n", mouse.xy.y);
}
}
/*
** Erase the world Bitmap from the old
** mouse position.
*/
bitblt(&world, world.rect, &display, MousePosition, F_XOR);

fprintf(stderr, "Your last words...\n");
lprintf("\nAbout to exit, any last words? (type now)\n");
lprintf("    Hit @ when ready to quit.\n");

/*
** Request the keyboard resource and wait
** until there are keyboard characters to
** be serviced.
*/
request(KBD);
wait(KBD);
```

Figure 7-1: Example Program - "jxmouse" (Sheet 3 of 4)

```
/*
** Save what is typed (up to MAXLEN characters)
** in the string pointed to by "p" until
** the user types '@'
*/
while ((c=(unsigned char)kbdchar()) != '@') {
    if (len < MAXLEN) {
        len++;
        *p++ = c;
    }

    /*
    ** Print character typed and then
    ** wait for more input from the keyboard.
    */
    lprintf("%c", c);
    wait(KBD);
}
*p = '\0';
/*
** Send the string to stderr.
*/
fprintf(stderr, "%s\n", words);

/*
** Must call exit for jx.
** stdout and stderr will be
** sent to the window
*/
exit();
}
```

Figure 7-1: Example Program - "jxmouse" (Sheet 4 of 4)

Chapter 8: Fonts and the Font Cache

“Font” and “Fontchar” Structures	8-1
“Font” Data Structure	8-1
“Fontchar” Data Structure	8-3
Drawing Characters on the Screen	8-5
Drawing Text Strings - “string”, “jstring”, and “strwidth”	8-8
Getting New Fonts from the Host - “getfont”	8-9
The 630 MTG Font Cache	8-12

“Font” and “Fontchar” Structures

Font is a data structure that describes a set of character images. **Fontchar** is a data structure that describes each character image in the **Font.bit** Bitmap. The data type definitions for **Font** and **Fontchar** are shown below and can be found in the file *DMD/include/font.h*. See Figures 8-1 (Font Bitmap) and 8-2 (Character Image Cell) for illustrations of the data fields in the **Font** and **Fontchar** data types.

“Font” Data Structure

The data type definition for **Font** is as follows:

```
typedef struct Font
{
    short n;
    char height;
    char ascent;
    long unused;
    Bitmap *bits;
    Fontchar info[1];
} Font;
```

The following list explains the different entries in the "Font Bitmap."

- n** Gives the number of character images in the font. **n** is also the ascii value of the last character image in the font.
- height** Gives the height in pixels of the Bitmap "bits".
- ascent** Gives the number of scan-lines from the top of the Bitmap "bits" to the baseline of the character images. All the character images are aligned vertically on the same baseline with some characters such as a **g** having a "descender" below the baseline.
- unused** Provides four extra bytes if needed.
- bits** This is the Bitmap containing the character images. It contains the bit pattern for each character arrayed adjacently into a long horizontal strip as illustrated below.

ABCDEFGHIJKLMNOPQRSTUVWXYZ

“Font” and “Fontchar” Structures

info[] This is an array of **Fontchar** data structures. **Font.info[n]** is a dummy Fontchar descriptor used for determining the right edge of the last character in the Bitmap "bits". The width of a "Character Image Cell" for the *i*'th character is

$$\text{Font.info}[i+1].x - \text{Font.info}[i].x$$

Figure 8-1 is an illustration of the **Font Bitmap**.

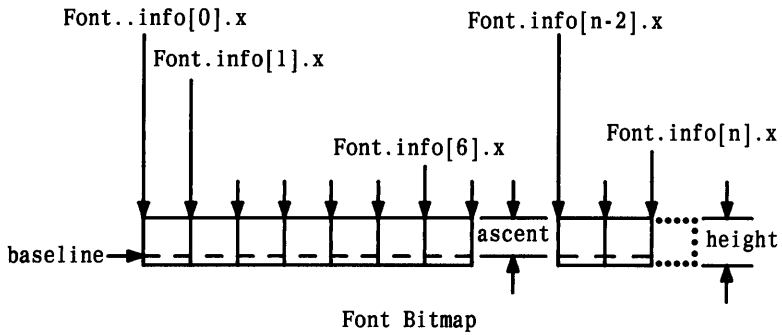


Figure 8-1: Font Bitmap

“Fontchar” Data Structure

The data type definition for `Fontchar` is as follows:

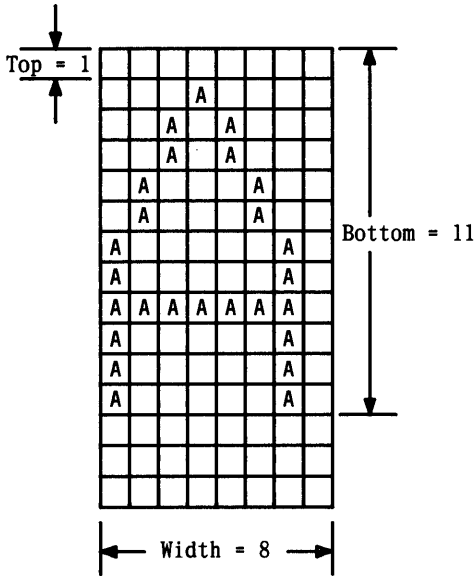
```
typedef struct Fontchar
{
    short x;
    unsigned char top;
    unsigned char bottom;
    char left;
    unsigned char width;
} Fontchar;
```

The following list explains the different entries in the "Character Image Cell."

- x** This is the pixel position of the left edge of the "Character Image Cell" in the Bitmap `Font.bits`.
- top** This is the first non-zero scan-line of the character image.
- bottom** This is the last non-zero scan-line of the character image.
- left** This is a signed 8-bit value used for kerning. See the example subroutine `drawchar` (Figure 8-3) for an example of the use of **left**.
- width** This is the width in pixels of the character image baseline. This is not necessarily equal to the width of the "Character Image Cell".

Figure 8-2 is an illustration of the "Character Image Cell."

“Font” and “Fontchar” Structures



Character Image Cell

Figure 8-2: Character Image Cell

Drawing Characters on the Screen

Drawing a character on the screen requires transferring the appropriate rectangle from the Font Bitmap to the correct location on the screen. For the F_STORE storage mode, the entire "Character Image Cell" must be copied from the Font Bitmap to the destination Bitmap. For the F_OR, F_XOR and F_CLR storage modes, the minimum enclosing rectangle of the character must be copied from the Font Bitmap to the destination Bitmap. When a character is displayed on the screen at a Point "p", the upper left-hand corner of the "character image cell" coincides with the Point "p".

Figure 8-3 gives the source code for the subroutine **drawchar** which will draw a character on the screen.

Drawing Characters on the Screen

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

/*Library routines and associated manual pages. */
void bitblt();          /* BITBLT(3R)      */

/*Library macros and associated manual pages. */
/* Point Pt()          Pt(3L)      */

main()
{}

void
drawchar(Character, FontPointer, DestinationBitmap,
          ScreenPoint, StorageCode)

char   Character;      /* character to be drawn */
Font   *FontPointer;   /* pointer to font used   */
Bitmap *DestinationBitmap; /* destination Bitmap    */
Point  ScreenPoint;   /* point to draw character */
Code   StorageCode;   /* graphics storage code  */
{
    Rectangle CharRect;
    Fontchar *CharDescriptor;

    /*
    ** Get the "Fontchar" character descriptor for "Character".
    */
    CharDescriptor = FontPointer->info+Character;

    /*
    ** Determine the enclosing rectangle for "Character"
    ** based on the graphical storage code used.
    */
}
```

Figure 8-3: Subroutine "drawchar" (Sheet 1 of 2)

```
if(StorageCode == F_STORE) {
    /*
    ** y-coordinates of the "character image cell".
    */
    CharRect.origin.y = 0;
    CharRect.corner.y = FontPointer->height;
} else {
    /*
    ** y-coordinates of the minimum enclosing rectangle.
    */
    CharRect.origin.y = CharDescriptor->top;
    CharRect.corner.y = CharDescriptor->bottom;
}

/*
** x-coordinates
*/
CharRect.origin.x = CharDescriptor->x;
CharRect.corner.x = (CharDescriptor+1)->x;

/*
** Display the character.
*/
bitblt(
/*Source Bitmap      */ FontPointer->bits,
/*Source Rectangle  */ CharRect,
/*Destination Bitmap*/ DestinationBitmap,
/*Destination Point */ Pt(ScreenPoint.x + CharDescriptor->left,
                          ScreenPoint.y + CharRect.origin.y),
/*Storage code      */ StorageCode);
}
```

Figure 8-3: Subroutine "drawchar" (Sheet 2 of 2)

Notice that the coordinate system places the origin of the tallest character at the specified point instead of at the baseline. This behavior is consistent with the coordinate system of **Rectangles** but requires some programming if characters from several fonts are placed on the same baseline.

Drawing Text Strings - “string”, “jstring”, and “strwidth”

The routines **string** and **jstring** draw null terminated text strings. These two functions are normally used instead of **bitblt** to draw characters onto the screen.

The **string** routine allows font selection for the text string. The parameter "p" accepted by **string** is a point that must be specified in screen coordinates. Two macros, **FONTWIDTH** and **FONTHEIGHT**, are available for usage with this function. **FONTWIDTH** returns the width of the space character in the given font. **FONTHEIGHT** returns the height of the given font. **strwidth** will calculate the width of a string based on the given font.

The **jstring** routine only draws strings with the **mediumfont** font. This function implicitly uses the window coordinate point, **PtCurrent**, as its starting point and returns the updated **PtCurrent**. The call, **jstrwidth**, is equivalent to **strwidth**.

For further details on these routines, see the manual pages **STRING(3R)**, **JSTRING(3L)**, and **STRWIDTH(3R)** in the *630 MTG Software Reference Manual*.

Getting New Fonts from the Host - “getfont”

getfont will download new fonts from the host for an application; this routine requires downloading the application with **jx**. See the manual page **INFONT(3R/3L)** for more details on **getfont**. **getfont** returns a pointer to a **Font** read from the specified UNIX System file. The following example program can be used to download a font from the *\$DMD/termfonts* directory and display a text string using the downloaded font. **argc** and **argv** are used in the program to pass the pathname of the font to be downloaded. A sample command line to download the program would be:

```
jx dmda.out $DMD/termfonts/12x25round
```

Figure 8-4 is an example program using **getfont**.

Getting New Fonts from the Host - "getfont"

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

/*Library routines and associated manual pages. */
Font *getfont();          /* INFONT(3L/3R)      */
Point sPtCurrent();      /* MOVETO(3L)    */

/* Global Variables */
Font *fp;

/*
** This program must be downloaded using jx.
** The pathname of the desired font is passed to
** the program as a command line argument using
** "argc" and "argv".
*/
main(argc, argv)
int argc;
char *argv[];
{
    lprintf("\n Loading font %s\n ", argv[1]);

    /*
    ** Download the font into 630 MTG memory.
    */
    fp = getfont(argv[1]);

    /*
    ** Display the string "This is a test."
    ** using the downloaded font.
    */
    string(fp, "This is a test.", &display, sPtCurrent(), F_STORE);

    /*
    ** Exit when the user types any key
    ** on the keyboard.
    */
    request(KBD);
    while(kbdchar() == -1) wait(KBD);
    exit();
}
```

Figure 8-4: Example Program Using "getfont"

Getting New Fonts from the Host - "getfont"

For more information on **getfont** and other routines that support **font** downloading, see the manual page **INFONT(3R/3L)** in the *630 MTG Software Reference Manual*.

The 630 MTG Font Cache

The 630 MTG Font Cache is dynamic storage for fonts in the 630 MTG terminal. In the previous example (Figure 8-4), only the application that downloads the font has access to the font. Fonts stored in the Font Cache are available for use by all applications running in the 630 MTG. The fonts **largefont**, **mediumfont**, and **smallfont** are stored in the 630 MTG's ROM and are permanent residents of the Font Cache. New Fonts can be added to the Font Cache by downloading them from the host into the 630 MTG's memory and issuing the command **fontcache** which places the specified font into the Font Cache. The command **fontremove** removes the specified font from the Font Cache and frees its memory.

Figure 8-5 shows how the previous example program can be modified to "cache" the downloaded font.

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

/*Library routines and associated manual pages. */
int fontcache();      /* FONTSAVE(3L)      */
Font *getfont();     /* INFONT(3L/3R)   */
Point sPtCurrent();  /* MOVETO(3L)      */

/* Global Variables */
Font *fp;

/*
** This program must be downloaded using jx.
** The pathname of the desired font is passed to the
** program as a command line argument using "argc" and "argv".
*/
main(argc, argv)
int argc;
char *argv[];
{
    lprintf("\n Loading font %s\n ", argv[1]);

    /*
    ** Download the font into 630 MTG memory.
    */
    fp = getfont(argv[1]);

    /*
    ** Place the font into the font cache.
    */
    fontcache(argv[1], fp);

    /*
    ** Display the string "This is a test."
    ** using the downloaded font.
    */
    string(fp, "This is a test.", &display, sPtCurrent(), F_STORE);

    /*
    ** Exit when the user types any key on the keyboard.
    */
    request(KBD);
    while(kbdchar() == -1) wait(KBD);
    exit();
}
```

Figure 8-5: Example Program - Caching a Downloaded Font

The 630 MTG Font Cache

After the test string is displayed, you will be able to go to one of the other windows on the terminal running the Windowproc terminal emulator and select the downloaded font for use in that window. This is possible since the Windowproc terminal emulator allows you to use any of the fonts that are currently in the 630 MTG Font Cache.

For further details on Font Cache management and usage, see the manual pages **FONTSAVE(3L)**, **FONTRREQUEST(3R)**, **FONTNAME(3R)**, and **FONTUSED(3R)** in the *630 MTG Software Reference Manual*.

Chapter 9: Interprocess Communications (Messages)

Introduction	9-1
What is a Message?	9-2
Creating a Message Queue	9-3
Sending and Receiving Messages	9-5
Sending a Message - "msgsnd"	9-5
Receiving a Message - "msgrcv"	9-6
Example Program - "messages1.c"	9-8
Message Queue Control	9-11
Example Program - "messages2.c"	9-14

Introduction

A form of Interprocess Communication (IPC) called "messages" is available on the 630 MTG. This feature is very similar to the message facility available on the UNIX System V Operating System.

A message contains information to be transferred from one application to another running in the 630 MTG. The message is posted by one application and then read by another. The transfer of messages must be prearranged by the applications.

Note: Message passing in the 630 MTG is a powerful facility; however, it is an advanced feature that most applications will not require.

What is a Message?

A message is a user-defined buffer whose first element is a message type identifier. The example shown below is defined in **message.h**.

```
typedef struct msgbuf {           /* a message */
    long mtype;                  /* message identifier */
    char mtext[1];              /* text of message */
} msgbuf;
```

The contents of a message is not limited to text only. The following is another example of a perfectly legitimate message:

```
typedef struct msgbuf2 {         /* a message */
    long mtype;                  /* message identifier */
    Rectangle msg_rect;         /* text of message */
    Point msg_pt;               /* more message text */
} msgbuf;
```

The message identifier **mtype** allows for unique identification of messages on a message queue. (This is described in more detail in the "Receiving a message - "msgrcv"" section of this chapter.) **mtype** must be greater than zero. The data fields that follow **mtype** are user definable and specify the content of the message.

Creating a Message Queue

Before any messages can be sent, a message queue must be created. The message queue is like a bulletin board on which applications can post messages. The message queue is created by the routine **msgget**. This routine is declared as follows:

```
long msgget(key, msgflg)
long key;
int msgflg;
```

The first argument, **key**, is a key to the message queue. When a message queue is created, a unique key and message queue identifier is associated with it. **msgget** translates a key into a message queue identifier that is used by all the other message routines. The key `IPC_PRIVATE` is guaranteed not to be used by any existing message queue.

The second argument, **msgflg**, is a bit vector that gives **msgget** further instructions on how to create the message queue. It should be noted that the queue may or may not already exist for the given key when **msgget** is called. The possible values of the **msgflg** are:

msgflg	Instructions
0	If a message queue with the given key already exists, return its identifier; otherwise fail (return -1).
<code>IPC_CREAT</code>	If a message queue with the given key already exists, return its identifier; otherwise, create a message queue for the key and return the new identifier.
<code>IPC_CREAT IPC_EXCL</code>	If a message queue with the given key already exists, fail (return -1); otherwise, create a message queue for the key and return the new identifier.

Creating a Message Queue

In addition to the bit vectors defined above, **msgflg** can be or'ed with **NO_SAVE**. If **NO_SAVE** is set when the message queue is created, the queue will be deleted when the application exits. Otherwise, the message queue will exist until **msgctl** (see the section "Message Queue Control") is called to remove queue or the terminal is powered down.

If successful, **msgget** returns a message queue identifier; otherwise, -1 is returned. The **msgget** routine can fail due to lack of memory. See the manual page **MSGGET(3L)** for more details.

Sending and Receiving Messages

Sending a Message - “msgsnd”

A message is added to a message queue by calling **msgsnd**. This routine is declared as follows:

```
int msgsnd(msqid, msgp, msgsz, msgflg)
long msqid;
struct msgbuf *msgp;
int msgsz, msgflg;
```

The first argument, **msqid**, is a message queue identifier as returned by the **msgget** routine. This identifies which message queue the message is to be added to.

The second argument, **msgp**, is the address of a message to be added to the queue. A copy of this message will be added to the queue unless **msgflg&NO_COPY** is true (see **NO_COPY** in the following table).

The third argument, **msgsz**, is the number of bytes in the message text. **msgsz** is the cumulative size in bytes of the data fields that follow the **mtype** field. The size can be 0.

The last argument, **msgflg**, is a bit vector. Possible values of **msgflg** and the associated instructions to **msgsnd** are as follows:

msgflg	Instruction
IPC_NOWAIT	If set, msgsnd returns immediately if it cannot put the message on the queue because of the queue size limit or a lack of memory. Otherwise, it will wait until there is room and available memory.

msgflg	Instruction
NO_COPY	This flag requires msgp to be a value returned from a call to alloc . Instead of copying the message into the queue, the message itself is put in the queue. The application will no longer own the memory pointed to by msgp .

See the manual page **MSGOP(3L)** for more details.

Receiving a Message - “msgrcv”

A message can be received and removed from a message queue by calling **msgrcv**. This routine is declared as follows:

```
int msgrcv(msqid, msgp, msgsz, msgtyp, msgflg)
long msqid, msgtyp;
struct msgbuf *msgp;
int msgsz, msgflg;
```

The first argument, **msqid**, is a message queue identifier as returned by **msgget**. This identifies from which queue the message is received.

The second argument, **msgp**, is the address where the message is copied. If the **NO_COPY** flag is set, this argument's meaning changes (see **NO_COPY** in the following table).

The third argument, **msgsz**, is the size of the message space pointed to by the second argument. If the message being received is greater than this size, **msgrcv** will fail unless the **MSG_NOERROR** flag is set (see **MSG_NOERROR** in the following table). If the **NO_COPY** flag is set, this field is ignored.

The fourth argument, **msgtyp**, specifies the type of message that is to be received. If **msgtyp** is zero, the first message on the queue is received. If **msgtyp** is greater than zero, the first message in the queue whose **mtype** equals **msgtyp** will be received. If **msgtyp** is less than zero, the first message of the lowest **mtype** is received if it is less than the absolute value of **msgtyp**.

The final argument, **msgflg**, is a bit vector. Possible values for **msgflg** are as follows:

msgflg	Instructions
IPC_NOWAIT	Do not wait in msgrcv . If this is not set, msgrcv will wait until a message of the type specified by msgtyp is in the queue.
MSG_NOERROR	If set and the message received is larger than msgsz , the message is clipped to msgsz . Otherwise, the message must be less than or equal to msgsz bytes for msgrcv to succeed.
NO_COPY	When set, the message is not copied into the buffer pointed to by msgp . Instead, the msgp argument is declared as: <pre>struct msgbuf **msgp;</pre> msgrcv will set *msgp to the address of the received message. The memory pointed to by *msgp can then be used as if it was created by a call to alloc .

Example Program - "messages1.c"

The program `messages1.c` in the `$DMD/examples/Messages` directory is a simple example of using messages. Figure 9-1 provides a printout of the source code for `messages1.c`. To use this program, compile `messages1.c` and download it into several windows. Whenever button 2 is depressed in one of these windows, a message will be sent. The window that receives the message will ring the bell and tell you it got the message.

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

/*Library routines and associated manual pages. */
void alarm();          /* RESOURCES(3R)      */
void exit();          /* EXIT(3R)           */
int kbdchar();        /* KBDCHAR(3R)       */
int local();          /* LOCAL(3R)          */
void lprintf();       /* LPRINTF            */
long msgget();        /* MSGGET(3L)        */
int msgrcv();         /* MSGOP(3L)         */
int msgsnd();         /* MSGOP(3L)         */
int request();        /* RESOURCES(3R)     */
void ringbell();      /* RINGBELL(3R)      */
void sleep();         /* SLEEP(3R)         */
int wait();           /* RESOURCES(3R)     */

/*library macros and associated manual page. */
/* int button2();          BUTTONS(3R)          */
/* int button3();          BUTTONS(3R)          */

/*
** buffer for message being sent/received.
*/
msgbuf mbuf;

main()
{
    /*
    ** message queue identifier from msgget
    */
    long msqid;
```

Figure 9-1: Example Program - "messages1.c" (Sheet 1 of 3)

```
/*
** Release the host connection.
*/
local();

lprintf("Use button 2 to send a message\n");

/*
** Request the needed resources.
*/
request(KBDMOUSEMSG);

/*
** Initialize message type.
** type must be > 0 to be valid
*/
mbuf.mtype = 1;

/*
** Create a message queue if it does not
** already exist. Exit if queue cannot
** be created.
*/
if((msqid = msgget(0x10000, IPC_CREAT)) == -1)
    exit();    /* couldn't get the queue */

/*
** Continue execution until the user
** hits a key on the keyboard.
*/
while(kbdchar() == -1)
{
    /*
    ** Release the CPU until one of the requested
    ** resources needs servicing.
    */
    wait(MOUSEKBDMSG);
}
```

Figure 9-1: Example Program - "messages1.c" (Sheet 2 of 3)

Example Program - "messages1.c"

```
/*
** Time to wake up.  Something must need service.
** First check to see if there are any messages on
** the queue.  If not, then check the mouse buttons.
** If that was not it then maybe the user hit a
** keyboard key.
*/
if(msgrcv(msqid, &mbuf, 1, (long)0, IPC_NOWAIT) != -1)
{
    ringbell();
    lprintf("I got a message.\n");
}
if(button2())
{
    /*
    ** Try to post a message when the user
    ** presses button 2.
    */
    if(msgsnd(msqid, &mbuf, 1, IPC_NOWAIT) == -1)
        lprintf("Failed to send.\n");
    else
        lprintf("Sent a message.\n");
    /*
    ** Go to sleep and let someone else read the
    ** message.
    */
    sleep(60);
}
if(button3())
{
    /*
    ** Release the mouse resource so that
    ** the control process can process
    ** button 3.
    */
    request(KBDMSG);
    sleep(2);
    request(MOUSE|KBDMSG);
}
}
```

Figure 9-1: Example Program - "messages1.c" (Sheet 3 of 3)

Message Queue Control

Another important data structure associated with messages is the message queue `msqid_ds`. From this, an application can find the status of the queue. `msqid_ds` contains information about who last used the message queue, when that was, and other important information. `msqid_ds` is defined in the `message.h` include file as follows:

```
typedef struct msqid_ds {
    Proc *   cid;
    short   msg_qnum;
    short   msg_qbytes;
    struct Proc * msg_lspid;
    struct Proc * msg_lrpid;
    unsigned long   msg_stime;
    unsigned long   msg_rtime;
    unsigned long   msg_ctime;
    message_list *msg_list;
    short   msg_curbytes;
    short   state;
    long   name;
    struct msqid_ds *next;
} msqid_ds;
```

The meanings of these fields are:

`cid` designates the owner/creator of the message queue.

`msg_qnum` is the number of messages currently in the message queue.

`msg_qbytes` is the number of bytes (used by messages) that the queue can hold before it is full.

`msg_lspid` designates the application that last put a message in the queue.

`msg_lrpid` designates the application that last read a message from the message queue.

Message Queue Control

msg_stime is the time that the last message was put in the message queue. The time is calculated by the routine "realtime".

msg_rtime is the time when a message was last received. Again, the time is calculated using "realtime".

msg_ctime is the time when this data structure was last changed. A change can occur because of a **msgsnd**, **msgrcv**, or **msgctl**.

msg_list points to the list of messages in the message queue.

msg_curbytes is the current total number of bytes (used by the messages) in the queue.

state holds the **NO_SAVE** flag. If this flag is set, the message queue will be deleted when the application designated by **cid** is terminated.

name is the key associated with the message queue.

next points to another message queue data structure. All the message queues are in a linked list.

The message queue data structure can be accessed by the **msgctl** routine. **msgctl** allows a program to look at the contents of the message queue data structure and set certain fields. It can also be used to delete message queues. **msgctl** is declared as follows:

```
int msgctl(msqid, cmd, buf)
long msqid;
int cmd;
struct msqid_ds *buf;
```

The first argument, **msqid**, is the message queue identifier.

The second argument, **cmd**, is a command that specifies an operation on the message queue. Available commands are:

cmd	Operation
IPC_SET	<p>Update selected fields of the specified message queue's data structure with that pointed to by buf. Those fields that can be set this way are:</p> <ul style="list-style-type: none">msg_qbytescidstate <p>If msg_qbytes is > MAX_QBYTES (the maximum allowable value), the command will fail. No other fields of the message queue data structure can be directly set by an application. This command also updates msg_ctime.</p>
IPC_STAT	<p>Copy the entire message queue data structure associated with the message queue identifier into the message queue data structure pointed to by buf.</p>
IPC_RMID	<p>Delete the message queue and any messages in the queue.</p>

The final argument, **buf**, is a pointer to another message queue data structure. This is the source for the **IPC_SET** command and the destination for the **IPC_STAT** command. See the manual page **MSGCTL(3L)** for more details.

Example Program - "messages2.c"

The program `messages2.c` in the `$DMD/examples/Messages` directory is a graphical demonstration of messages. Figure 9-2 provides a printout of the source code of `messages2.c`. To use this demo, compile `messages2.c` and download it into several windows. It will begin shooting square bullets toward the center of all the other windows running the example program.

The `messages2.c` example demonstration performs the following functions. A global message queue with the key `0x44454d4f` (ASCII for "DEMO") is created. Each instance of the `messages2.c` application then sends a message to the global message queue. Each message contains the coordinates for the center of its respective window. This information is then used by all the other windows to determine the direction to shoot bullets.

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

/*
** Constant definitions.
*/
#define MAX_TARGET 20          /* maximum number of targets */
#define MSG_TYPE (long)1     /* message type on global queue */
#define MSG_SIZE 8           /* size of my message */
#define GLB_Q_KEY 0x44454d4f /* Key for Global Queue is DEMO */
#define SS 3                 /* size of a shot */
#define FAILURE -1          /* message get failure */

/*
** Type Definitions.
*/
typedef struct MyMsgBuf {
    long mtype;
    Point WindowCenter;
    long LocalMsgQid;
} MyMsgBuf;
```

Figure 9-2: Example Program - "messages2.c" (Sheet 1 of 9)

```
typedef struct Shot {
    Point ShotCenter;
    Point Velocity;
} Shot;

/*
** Global Variables.
*/
Point Target[MAX_TARGET];
int NumOfTargets;

/*Library routines and associated manual pages.      */
Point add();          /* PTARITH(3R)      */
Point div();          /* PTARITH(3R)      */
int  kbdchar();       /* KBDCHAR(3R)      */
int  local();         /* LOCAL(3R)        */
long msgget();        /* MSGGET(3L)       */
int  msgctl();        /* MSGCTL(3L)       */
int  msgsnd();        /* MSGOP(3L)        */
int  msgrcv();        /* MSGOP(3L)        */
int  request();       /* RESOURCES(3R)    */
void sleep();         /* SLEEP(3R)        */
Point sub();          /* PTARITH(3R)      */

/* Local function declared in this file */
void AddToList();
void DrawShots();
void Failure();
Point GetWindowCenter();
void Shoot();
```

Figure 9-2: Example Program - "messages2.c" (Sheet 2 of 9)

Example Program - "messages2.c"

```
main()
{
    MyMsgBuf MyMsg, TempMsg;
    long GlbMsgQid;
    msqid_ds LocalMsgQ;
    int i;

    /*
    ** This application does not need a host connection.
    */
    local();

    /*
    ** Request the use of the Keyboard.
    */
    request(KBD);

    if((GlbMsgQid = msgget(GLB_Q_KEY, IPC_CREAT)) == FAILURE)
        /*
        ** Could not open global message queue.
        */
        Failure("Can't get global message queue");

    MyMsg.LocalMsgQid = msgget((long)IPC_PRIVATE, IPC_CREAT|NO_SAVE);
    if(MyMsg.LocalMsgQid == FAILURE)
        /*
        ** Could not open my local message queue.
        */
        Failure("Can't create local message queue");

    MyMsg.mtype = MSG_TYPE;
    MyMsg.WindowCenter = GetWindowCenter();
    if(msgsnd(GlbMsgQid, &MyMsg, MSG_SIZE, IPC_NOWAIT))
        /*
        ** Could not put my message into the global message queue.
        */
        Failure("Can't send my first message");
}
```

Figure 9-2: Example Program - "messages2.c" (Sheet 3 of 9)

```
/*
** Main loop of program.
*/
while (kbdchar() != 'q') {
    if (msgctl(MyMsg.LocalMsgQid, IPC_STAT, &LocalMsgQ))
        /*
        ** Someone has removed my local message queue.
        ** They must want me dead.
        */
        Failure("Someone wants me dead");

    if (msgctl(GlbMsgQid, IPC_STAT, &LocalMsgQ))
        /*
        ** Someone has removed the global message queue.
        */
        Failure("global queue is gone");

    /*
    ** Get number of messages currently in the
    ** global message queue.
    */
    i = LocalMsgQ.msg_qnum;
    NumOfTargets = 0;

    /*
    ** If my window was reshaped, erase the window and
    ** recalculate the center of the window.
    */
    if(P->state & RESHAPED) {
        if(P->state & MOVED)
            rectf(&display, Drect, F_CLR);
        P->state &= ~RESHAPED;
        MyMsg.WindowCenter = GetWindowCenter();
    }

    /*
    ** Read all the messages off the global message queue.
    ** If the application that sent the message still exists,
    ** then add that process to the list of targets and put
    ** that applications message back on the global queue.
    ** Otherwise just throw the message away and read the
    ** next message from the queue.
    */
}
```

Figure 9-2: Example Program - "messages2.c" (Sheet 4 of 9)

Example Program - "messages2.c"

```
while(i-- &&
    msgrcv(GlbMsgQid, &TempMsg,
          MSG_SIZE, MSG_TYPE, IPC_NOWAITMSG_NOERROR) != -1) {
    if(msgctl(TempMsg.LocalMsgQid, IPC_STAT, &LocalMsgQ) == 0) {
        /*
         ** Application still exists.
         */
        if(TempMsg.LocalMsgQid != MyMsg.LocalMsgQid)
            /*
             ** If it is not my own message,
             ** then add it to the list of targets.
             */
            AddToList(TempMsg.WindowCenter);
        else
            /*
             ** I just read my own message.
             ** Make sure my window center is up
             ** to date.
             */
            TempMsg.WindowCenter = MyMsg.WindowCenter;

        /*
         ** Put the message back on the global queue.
         */
        if (msgsnd(GlbMsgQid, &TempMsg, MSG_SIZE, IPC_NOWAIT))
            /*
             ** Oh Oh! The message send has failed.
             ** Remove the queue that failed.
             */
            msgctl(TempMsg.LocalMsgQid, IPC_RMID, &LocalMsgQ);
    }
}
```

Figure 9-2: Example Program - "messages2.c" (Sheet 5 of 9)

```
    /*
    ** If i is greater than or equal to zero, then for some reason
    ** we were not able to read all the messages from the global
    ** message queue. Better quit.
    */
    if (i >= 0)
        Failure("I can't read from the global queue");

    /*
    ** Shoot bullets at the other windows running this application.
    */
    Shoot();

    /*
    ** In case the number of targets was zero, switch out
    ** here to let other applications run
    */
    sleep(1);
}

void
AddToList(p)
Point p;
/*
** Add a target to the list.
*/
{
    if (NumOfTargets == MAX_TARGET)
        /*
        ** Over the maximum target limit.
        */
        return;
}
```

Figure 9-2: Example Program - "messages2.c" (Sheet 6 of 9)

Example Program - "messages2.c"

```
/*
** Add target to the list
*/
Target[NumOfTargets++] = p;
}

void
Shoot()
{
    int i, j;
    Shot shot[MAX_TARGET];

    for (i=0; i<NumOfTargets; i++) {
        shot[i].ShotCenter = GetWindowCenter();
        shot[i].Velocity =
            div(sub(Target[i], shot[i].ShotCenter), 50);
    }

    DrawShots(shot);
    for (j=0; j<50; j++) {
        sleep(5);
        if (P->state & RESHAPED)
            return;

        /*
        ** Erase the old shots.
        */
        DrawShots(shot);

        /*
        ** Update the center position of each shot.
        */
        for (i=0; i<NumOfTargets; i++)
            shot[i].ShotCenter =
                add(shot[i].ShotCenter, shot[i].Velocity);
    }
}
```

Figure 9-2: Example Program - "messages2.c" (Sheet 7 of 9)

```
        /*
        ** Draw the new shots.
        */
        DrawShots(shot);
    }

    /*
    ** Erase all shots.
    */
    DrawShots(shot);
}

void
DrawShots(shot)
Shot *shot;
{
    int i;

    for(i=0; i<NumOfTargets; i++)
        /*
        ** Draw the shot in the display Bitmap.
        */
        rectf(&display,
            Rpt(sub(shot[i].ShotCenter, Pt(SS,SS)),
                add(shot[i].ShotCenter, Pt(SS,SS))),
            F_XOR);
}

void
Failure(string)
char *string;
{
    /*
    ** Print error message and wait for user to hit q.
    */
    jstring(string);
    request(KBD);
    while (kbdchar() != 'q') wait(KBD);
    exit();
}
```

Figure 9-2: Example Program - "messages2.c" (Sheet 8 of 9)

Example Program - "messages2.c"

```
Point
GetWindowCenter()
{
    /*
    ** Calculate the center Point of the window
    */
    return add(div(sub
(Direct.corner, Direct.origin), 2), Direct.origin);
}
```

Figure 9-2: Example Program - "messages2.c" (Sheet 9 of 9)

Chapter 10: Application Caching

Introduction	10-1
Caching an Application	10-2
Explicit Arguments - "s" and "f"	10-2
Use of "argv[0]"	10-3
Implicit Arguments	10-3
Return Value of Cache	10-4
Example of Caching an Application	10-4
Notes on Caching an Application	10-5
Restarting the Application Using the Download Command	10-5
Restarting the Application Using the "More" Menu	10-5
Caching a Host-Connected Application	10-6
Caching a Local or Connected Application	10-6
Caching a Shared Application (A_SHARED)	10-6
Removing Applications from the Cache	10-8
Reshapability of Cached Applications	10-9
Non-Reshapable Application (NO_RESHAPE)	10-9
Default Window Outline (P->ctob)	10-9
Non-Reshapable, Default Window Outline (NO_RESHAPE and P->ctob)	10-10
Cached Applications and ".text," ".data," and ".bss" Sections	10-11
Sharing the ".data" Section	10-12
Modifying the ".data" Section	10-12
Sharing the ".bss" Section	10-12
Initializing the ".bss" Section	10-13
Saving the ".bss" Section	10-13
Writing Shared Text Applications	10-14

Chapter 10: Application Caching

- Modifying Global Variables 10-14
- Eliminating Global Variables 10-14
 - Method #1 - Use Only Local Variables 10-14
 - Method #2 - Use Local Variables and Constant Global Variables 10-14
 - Method #3 - Dynamic Allocation of Global Variables 10-16
- Porting Existing Applications to Run Shared Text 10-18
 - Example of Porting an Existing Application to Run Shared Text 10-18
- Final Notes on Shared Text Applications 10-19

- Example Programs for Application Caching 10-21

Introduction

The Application Caching Facility allows the 630 MTG programmer to cache an application after it is downloaded. Caching an application causes the 630 MTG's operating system to retain the application's code and data in the 630 MTG's memory so that the application, once terminated, can be restarted without being downloaded again. Without application caching, any application that is deleted by the user or exits by itself can only be restarted by another download from the host.

During execution, an application places itself in the 630 MTG application cache by calling the routine **cache**. (See the manual page **CACHE(3L)** for more information.) When a cached application is terminated (through deletion or exit), it can be restarted either by executing the download command that was used initially or by selecting the application from the Button 3 **More** menu. In either case, the application restarts without another download.

Note: In the following discussion on application caching, several example programs have been provided for demonstration purposes. The source code for these examples is provided in *\$DMD/examples/Caching*. Also, a printout of each example program (Figures 10-1 through 10-17) is provided in the "Example Programs for Application Caching" section at the end of this chapter.

Caching an Application

The routine `cache`, once executed, causes the code and data for the application to be placed in the application cache. `cache` is declared as follows:

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

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

Applications calling the `cache` routine must include the file `object.h`.

Explicit Arguments - "s" and "f"

The first argument, "s", is a pointer to a null terminated character string that is used as the entry in the Button 3 **More** menu. If "s" is a null pointer, then the character string pointed to by `argv[0]`, stripped of any path name prefix, will be used as the **More** menu entry.

The second argument, "3", is a bit vector composed of the bitwise inclusive "OR" of zero or more of the following predefined masks:

A_SHARED

Allow multiple instances of the application to run simultaneously. If this bit is not set, only a single instance of the cached application will be allowed to run at a time. Setting `A_SHARED` forces `A_DATA` and `A_BSS` to be set.

A_NO_SHOW

Do not advertise the application in the Button 3 **More** menu. If not set, the string pointed to by "s" will be placed in the **More** menu. If "s" is null, the string pointed to by `argv[0]` will be placed in the **More** menu.

A_BSS Do not reset to zero the uninitialized global and static variables (`.bss` section) on subsequent startup of the cached application. If not set, all these variables will be reset to zero. (See the section entitled "Cached Applications and `.text`, `.data`, and `.bss` Sections" in this chapter.)

A_DATA

Do not reset initialized global and static variables to their original values (**.data** section) on subsequent startup of the cached application. If not set, a "snapshot" of these variables is taken when the function **cache** is called. This snapshot is used to restore initialized variables to their original values on subsequent startup of the application. (See the section entitled "Cached Applications and **.text**, **.data**, and **.bss** Sections" in this chapter.)

A_NO_BOOT

Do not allow the cached application to be started using **dmdld**. If not set, the cached application can be restarted in a new window using **dmdld**.

A_PERMANENT

Do not allow the application to be removed from the cache.

Use of "argv[0]"

When an application is downloaded using **dmdld**, **argv[0]** is set to the name of the executable object that is downloaded. When an application calls **cache**, **argv[0]** (stripped from any pathname prefix) is used as the tag for the application in the 630 MTG application cache. When any application is downloaded using **dmdld**, this tag is searched for in the 630 MTG application cache. If the tag is found in the application cache, then the application is immediately restarted without another download.

Note: Remember, the application is tagged by **argv[0]** and not the string pointed to by the argument "s". "s" is only used to specify an alternate string to put in the **More** menu.

Implicit Arguments

Along with the explicit arguments to the **cache** routine, there are implicit arguments as well. **cache** implicitly determines the need for a host connection, window reshaping, and default window size from the current state of the application. The implications of the explicit and implicit arguments will be pointed out by the example programs in this chapter. The source code for all the examples may be found in the directory *\$DMD/examples/Caching*.

Return Value of Cache

If the calling application is successfully cached, the **cache** routine will return a 1. Otherwise, a 0 is returned. **cache** will fail if there is already an application in the cache with the same tag or there is not enough memory in the terminal to cache the application.

Example of Caching an Application

Figure 10-1 gives the source code for **cache1.c** which is a very simple example of application caching.

Note: The function call to **local** will fail if your application is running in the only window connected to the host. Therefore, the example programs in this section should only be downloaded when there is more than one window connected to the host.

In **cache1.c**, the routine **cache** is called with the argument "s" set to null and the argument "f" set to zero. This specifies the following default set of instructions on how to cache the application:

1. Use the string pointed to by **argv[0]**, stripped of any pathname prefix, to advertise the application in the **More** menu.
2. Allow only one instance of the application to be invoked from the cache.
3. If there are uninitialized global variables, that is, variables in the **.bss** section, set them to zero every time the application is restarted.
4. Take a snapshot of the application's **.data** section and restore the **.data** section to this snapshot every time the application is restarted.

Use the following command lines to compile and download the example application:

```
dmdcc -o cache1 cache1.c
dmdld cache1
```

Note: For more information on **dmdcc** and **dmdld**, refer to the appropriate manual pages in the *630 MTG Software Reference Manual*.

Notes on Caching an Application

Once the compiled version of `cache1.c` (`cache1`) is downloaded, execution begins, a local window is created, the string "Hello World" is printed, and the application waits for you to hit any key before it exits. Hitting any key terminates the application.

Restarting the Application Using the Download Command

`cache1` can be restarted without doing a second download by creating a new window and issuing the download command again:

```
dmdld cache1
```

Since the code and data for `cache1` is in local memory, `cache1` will begin execution immediately in the new window.

Note: Before restarting `cache1` with `dmdld`, you must terminate the instance of `cache1` that is currently running. Otherwise, if an instance of `cache1` already exists, `dmdld` will be told that `cache1` is not available and, therefore, it will redownload `cache1`. This is true for all non-shared applications.

Once the application has been cached, you no longer have to be in the same directory as the application's executable object to restart the application with the download command. Once again, hitting any key terminates the example application `cache1`.

Restarting the Application Using the "More" Menu

In the button 3 **More** menu, notice that `cache1` is a listed menu item. Selecting `cache1` from the menu produces a sweep cursor that is a prompt to sweep out a new window. Once the window has been created, `cache1` will begin execution in the new window.

Note: If you select `cache1` in the **More** menu while an instance of `cache1` is running, the `cache1` window will be made top and current.

Caching a Host-Connected Application

The code for `cache2.c` is the same as `cache1.c` except that the call to `local` has been removed.

Compile and download `cache2.c` in the same manner as shown for the Example Program `cache1.c`. Figure 10-2 gives the source code of `cache2.c`.

Once execution of `cache2.c` begins, the border of the window remains solid since `cache2` retains its host connection. Notice the difference in the **More** menu item for `cache2`. The `cache2` menu item has a **Host** submenu that allows you to choose which host you would like `cache2` to be connected to when it is restarted. Note, however, that the host submenu will only be present when there is no window currently running `cache2`. This is because the `cache` routine was instructed, by default, to allow the cached application to run in only one window at a time. If `cache2` is already running in a window, selecting the `cache2` menu item will make that window top and current.

Caching a Local or Connected Application

`cache3.c` demonstrates that the `cache` routine uses the current state of the Host connection as an implicit argument. Compile and download `cache3.c`. Figure 10-3 gives the source code for `cache3.c`.

The difference in `cache3.c` from `cache2.c` is that the `cache` routine is called before `local` in `cache3.c`. Since `cache3` still has a host connection when the `cache` routine is called, the `cache3` menu item will be given a **Host** submenu. This means that when you restart `cache3` from the **More** menu, you will need to select a host even though the application releases its host connection by calling `local`. Therefore, if your application does not need a host connection, it should call `local` before `cache`.

Caching a Shared Application (A_SHARED)

The first three examples allowed only one window to run the cached application at a time. `cache4.c` demonstrates how a cached application can be shared by multiple windows simultaneously. Compile and download `cache4.c`. Figure 10-4 gives the source code for `cache4.c`.

Specifying `A_SHARED`, tells **cache** that this application can have multiple instances running simultaneously. Note that each instance of the application will be running from the same code in memory; that is, a new copy of the code is not created for each instance. This is commonly called a "shared text" application. Further implications of "shared text" applications will be discussed in the section entitled "Writing Shared Text Applications" of this chapter.

After downloading **cache4** and before terminating its execution, select **cache4** from the **More** menu and sweep a new window. Note that there are now two instances of **cache4** running. You can continue to create instances of **cache4** by selecting it from the **More** menu or typing `dmdld cache4` in a new window.

Removing Applications from the Cache

The application **ucache** in the directory *\$DMD/bin* allows you to remove cached applications to free up memory and clean up your **More** menu. **ucache** is a downloadable application that caches itself and allows you to selectively remove cached applications. The **ucache** menu item has an associated submenu listing all the applications currently in the application cache. Items in this submenu that are not "greyed" can be selected for removal from the cache. Greyed items in the menu are either permanent members of the cache, such as **PF Edit** and **Setup**, or they are cached applications that are currently in use.

Reshapability of Cached Applications

The following examples (`cache5.c`, `cache6.c` and `cache7.c`) demonstrate how the `NO_RESHAPE` bit in the process state variable (`P->state`) and the character-to-bits function (`P->ctob`) affect the reshapability of a cached application.

Non-Reshapable Application (`NO_RESHAPE`)

`cache5.c` demonstrates how you can make your cached application non-reshapable. Figure 10-5 gives the source code of `cache5.c`.

`cache5.c` is the same as `cache1.c` except for the fact that the `NO_RESHAPE` bit is set in the process state variable (`P->state`). When `cache` detects that `NO_RESHAPE` is set, it records the current size of the application's window and uses that as the default window size when the application is restarted. When you start `cache5` from the **More** menu, you get a default window outline and no sweep cursor. Notice that if you use `dmdld cache5` to restart `cache5` in a window, the window will automatically be reshaped to the default window size.

Default Window Outline (`P->ctob`)

`cache6.c` demonstrates the use of `P->ctob` to specify a default window outline. Figure 10-6 contains the source code for `cache6.c`.

Note that `P->ctob` is not specific to caching only. It can be used by any application to specify a default window size for **Reshape**. See the manual page **BTOC(3R)** in the *630 MTG Software Reference Manual*.

`cache6.c` is the same as `cache1.c` except the pointer `P->ctob` is initialized to point to the function `WindowSize`. `WindowSize` is called indirectly through `P->ctob` by the 630 MTG operating system whenever the application's window is reshaped. It is also called by `cache` after the initial download of the application to determine a default outline to display when the application is restarted from the **More** menu.

Reshapability of Cached Applications

The Point that **WindowSize** returns specifies the width and height of the default window outline. When an instance of **cache6** is started from the **More** menu, you will see a sweep cursor and a default window outline. Clicking Button 3 selects the default window. You can also sweep a window of any size if you desire. This is the same way that **Setup** works when invoked from the **More** menu.

Non-Reshapable, Default Window Outline (NO_RESHAPE and P->ctob)

cache7.c demonstrates what happens when both the "NO_RESHAPE" bit is set and **P->ctob** is initialized. Figure 10-7 contains the source code for **cache7.c**.

When **cache7.c** is initially downloaded into a window, that window will become non-reshapable regardless of what size it is. When **cache7** is invoked from the **More** menu, however, a default outline (as specified by the **WindowSize** function) is displayed without a sweep cursor. Clicking Button 3 creates the window. The window will always default to the size specified by **WindowSize** and will not be reshapable. This is the same way that **PF Edit** works when invoked from the **More** menu.

Cached Applications and “.text,” “.data,” and “.bss” Sections

The **cache** routine treats an application as an object with three sections:

- .text** This section contains the executable instructions of the application.
- .data** This section contains all "initialized" external and "initialized" static variables.
- .bss** (Blank Storage Segment) This section contains all "uninitialized" external and "uninitialized" static variables.

The **cache** routine saves these sections in the terminal's memory. All instances of the same cached application share these three sections.

For example, consider the following program:

```
#include <dmd.h>

int temp1;
int temp2 = 0;

main ()
{
    static int temp3;
    static int temp4 = 0;

    foo ();
}

foo()
{
}
```

The globals **main()** and **foo()** are in the **.text** section. The globals **temp1** and **temp3** are respectively uninitialized external and uninitialized static variables, thus they are in the **.bss** section. The globals **temp2** and **temp4** are respectively initialized external and initialized static variables, thus they are in the **.data** section.

The discussion that follows describes some of the implications of sharing these sections.

Sharing the “.data” Section

Modification of variables in the `.data` section is common in many applications. By default, a cached application’s `.data` section will be restored every time the application is restarted to the initial snapshot that was taken when `cache` was called. If, however, you set the `A_DATA` bit in the argument “f”, the `.data` section of your application will NOT be restored upon subsequent invocations.

Modifying the “.data” Section

`cache8.c` demonstrates the feature described above. Figure 10-8 contains the source code for `cache8.c`.

Normally, you would expect each instance of `cache8` to produce the following two lines of output:

```
temp1 = 1, temp2 = 2
temp1 = 2, temp2 = 3
```

This, however, is true only the first time the application is run. The second time the application is invoked from the cache, the output will be:

```
temp1 = 2, temp2 = 3
temp1 = 3, temp2 = 4
```

The third restart will produce:

```
temp1 = 3, temp2 = 4
temp1 = 4, temp2 = 5
```

Subsequent restarts of `cache8` will follow the same pattern. This is because `cache8` modifies the contents of its `.data` section, which is not reinitialized on restart.

Sharing the “.bss” Section

Uninitialized external and static variables end up in the `.bss` section of your application. The standard C language has a rule governing the default initialization of uninitialized variables that states:

Cached Applications and “.text,” “.data,” and “.bss” Sections

"In the absence of explicit initialization, external and static variables are guaranteed to be initialized to zero."

Unless otherwise specified, this rule is maintained for cached applications. The `.bss` section of a cached application is cleared every time the application is invoked unless `cache` is called with the "A_BSS" flag set. The two following examples (`cache9.c` and `cache10.c`) demonstrate this feature.

Initializing the “.bss” Section

`cache9.c` demonstrates that uninitialized external and static variables are set to zero every time a cached application is restarted. Figure 10-9 contains the source code for `cache9.c`.

No matter how many times you restart `cache9`, you will get the following two lines of output because the `.bss` section is set to zero:

```
temp1 = 0, temp2 = 0
temp1 = 1, temp2 = 1
```

Saving the “.bss” Section

`cache10.c` demonstrates how the A_BSS flag causes the `.bss` segment to remain static. The source code for `cache10.c` is shown in Figure 10-10.

The output from `cache10.c` will be very similar to that of `cache8.c`. Initially `cache10` will produce the following output:

```
temp1 = 0, temp2 = 0
temp1 = 1, temp2 = 1
```

When restarted, `cache10` will produce the output:

```
temp1 = 1, temp2 = 1
temp1 = 2, temp2 = 2
```

When restarted again:

```
temp1 = 2, temp2 = 2
temp1 = 3, temp2 = 3
```

This sequence of output occurs because the global variable `temp1` and the static variable `temp2` both reside in the `.bss` section of the code, and that section is not being initialized to zero each time `cache10` is restarted.

Writing Shared Text Applications

A shared text application has the following properties:

Multiple instances can exist simultaneously and each instance shares a single copy of the `.text`, `.data`, and `.bss` sections.

Most applications can be written to run in a shared text environment. To accomplish this, however, you must use some special programming techniques to make sure that one instance of the application does not modify the context of another instance of the application.

Modifying Global Variables

The most common context modification problem occurs when one instance of the application modifies a global variable that another instance depends on. This is a problem since all instances of the application share the same `.bss` and `.data` section. Eliminating this problem is the key to writing shared text applications. In the majority of cases, most, if not all, global variables must be removed from an application to run properly in a shared text environment.

Eliminating Global Variables

The following discussion presents three different methods that you can use to eliminate global variables from your application.

Method #1 - Use Only Local Variables

The simplest method for eliminating global variables from your application is just not to use any. This means that all variables in your application must be local to each function in the program. This approach enhances modularity but can make programming very difficult.

Method #2 - Use Local Variables and Constant Global Variables

A slightly less restrictive method is to use global variables that remain constant during the execution of your application, such as initialized external and static variables, and place all dynamic global variables into a data structure that is declared locally in the routine `main`.

Consider the simple application shown below:

```
#include <dmd.h>

int global1, global2;

main()
{
    global1 = global2 = 0;
    lprintf("global1 = %d, global2 = %d", global1, global2);
    global1++;
    global2++;
    lprintf("global1 = %d, global2 = %d", global1, global2);
    for (;;) wait (CPU);
}
```

If this application is to operate properly in a shared text environment, then the two global variables **global1** and **global2** must be eliminated. This is accomplished in the cached application shown in Figure 10-11.

cache11.c places all dynamic global variables into a data structure that is local to **main**. Local variables are always placed in the application's stack memory.

Note: Stack memory is not initialized to zero; therefore, variables placed on the stack must be initialized before they are used.

Information on Parameter Passing

If there is a need for a subroutine to access the variables in a local data structure, such as "myglobals" in **cache11.c**, then one of the parameters in the subroutine call must be the address of the data structure as illustrated in the example program **cache12.c**. The source code for **cache12.c** is given in Figure 10-12.

Having all dynamic variables in a data structure local to **main** works out well if the size of the data structure does not get too large (greater than 1K bytes) and use up too much of your application's stack memory. When the data structure becomes too large, you can either dynamically allocate the data structure (see Method #3) or increase the stack size (see the manual pages **dmdcc(1)** and **dmdld(1)**).

Method #3 - Dynamic Allocation of Global Variables

When the size of the data structure containing your application's dynamic global variables becomes too large, you will want to dynamically allocate this data structure rather than making it a local variable in **main**. The example program **cache13.c**, shown in Figure 10-13, demonstrates this method.

Note: Dynamically allocated memory is initialized to zero.

For more information on dynamic memory allocation, see the manual page **ALLOC(3R)**.

Additional Information on Parameter Passing - "P->appl"

cache13.c demonstrated one method of parameter passing for the data structure containing your dynamic global variables. However, this method of parameter passing is not always possible. For example, consider the **dfn**, **hfn**, and **bfm** routines of **tmenuhit**. One may wish to have the routines access data specific to one instance of an application. Normally, this would be done with global variables. However, **cache13.c** has no global variables; furthermore, the 630 MTG operating system has predefined the parameter with which **dfn**, **hfn**, and **bfm** are called.

For this reason a special field called **appl** has been provided in the application's "process structure" and is unique for each application. **P->appl** can be accessed globally and initialized to point to the data structure that contains your global variables. In this way, all routines in your application would be able to access your global variables through **P->appl**. The example program, **cache14.c** (Figure 10-14), demonstrates this method of parameter passing.

cache14 works very well but the source code is cumbersome to write because of all the type casting that is necessary to access the global variables. This problem can be overcome, however, by using **#defines** as demonstrated in **cache15.c**. The source code for **cache15.c** is given in Figure 10-15.

Notice the use of the underscore character "_" in the naming of the fields in the "Globals" data structure. The reason for this is to make sure that the variable name in the "Globals" data structure differs from the name used in the **#define**. If you were to declare your global structure and **#defines** as follows:

```
typedef struct Globals {
    int global1;
    int global2;
} Globals;

#define global1      (((struct Globals *) (P->app1))->global1)
#define global2      (((struct Globals *) (P->app1))->global2)
```

you would get a compiler error that says something like: "D_Ref: too much pushback." This problem occurs because we have created a recursive macro definition.

Warnings About Using "P"

You must be careful about the use of the variable "P" in certain routines, especially those routines defined in your application that are called by one of the 630 MTG system processes. For example, the routine pointed to by **P->ctob** is called by the 630 MTG control process. Refer to the manual page **BTOC(3R)**. Even though the routine pointed by **P->ctob** is defined in your application, you cannot use the variable "P" to reference your application's process structure. This is because "P" will be pointing to the process structure of the 630 MTG control process when **P->ctob** is called. Therefore, when the control process calls **P->ctob**, it passes a pointer to your application's process structure as a parameter. The example program, **cache16.c** (Figure 10-16), demonstrates the use of the **P->ctob** routine when it must access your "Globals" data structure.

In **cache16.c**, **P->ctob** and the variables **width** and **height** must be initialized before **cache** is called. This is because the first time **cache** is called, (the first time that the application is run after the initial download) **cache** will call **P->ctob** to determine what default outline to display for subsequent invocations of the cached application.

Porting Existing Applications to Run Shared Text

To port an existing application to the shared text environment, declare a data structure to hold all the dynamic global variables of the application as demonstrated in the previous examples. Then, dynamically allocate the data structure and set **P->appl** to point to the allocated structure. Near the beginning of your source code or in an include file, declare the type of your global data structure and **#defines** as shown below:

```
typedef struct Global {
    type1      _global1;
    type2      _global2;
    .
    .
    .
} Global;

#define global1  (((struct Global *) (P->appl))->_global1)
#define global2  (((struct Global *) (P->appl))->_global2)
.
.
.
```

This will make all existing references to the global variables in your application go through the **P->appl** indirection.

Example of Porting an Existing Application to Run Shared Text

As a final example of a shared text application, the example program **message2.c** from the previous chapter "Interprocess Communications (Messages)" has been ported to cache itself as a shared text application. The source code for this example can be found in the directory *\$DMD/examples/Caching* under the name **cache17.c**. Figure 10-17 gives the source code for **cache17.c**.

Final Notes on Shared Text Applications

Up to this point, the sharing of global variables between all instances of a cached application has been discouraged. However, this feature of sharing global variables can be used to your advantage in certain cases. Because global variables are shared by all instances of the application, they can be used as a form of inter-process communication: one instance of the application can set a variable which another instance is watching.

There is a subtle feature of the 630 MTG operating system that makes the sharing of global variables between instances of a cached application feasible. This has to do with the way applications are scheduled to run. A cached application is guaranteed that a shared global variable will not be changed by another instance of the application as long as it does not release the CPU.

Warning: You must be careful when using this feature because some of the 630 MTG library routines, such as `sendnchars`, release the CPU for you.

For further details about caching, see the manual pages `CACHE(3L)`, `DECACHE(3L)`, `CMDCACHE(3L)`, and `UCACHE(1)` in the *630 MTG Software Reference Manual*.

Example Programs for Application Caching

The source code for the example programs on application caching is included in this section (Figures 10-1 through 10-17).

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

/* List of routines and associated manual page */
int cache();          /* CACHE(3L) */
int kbdchar();       /* KBDCHAR(3R) */
int local();         /* LOCAL(3R) */
void lprintf();      /* PRINTF(3L) */
int request();       /* RESOURCES(3R) */
int wait();          /* RESOURCES(3R) */

main()
{
    /*
    ** Release the host connection.
    */
    local();

    /*
    ** Put application in the cache.
    */
    cache((char *)0, 0);

    lprintf("\n Hello World.");

    /*
    ** Wait for the user to hit a key.
    */
    request(KBD);
    while (kbdchar() == -1) wait(KBD);
}
```

Figure 10-1: Example Program "cache1.c"

Example Programs for Application Caching

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

/* List of routines and associated manual page */
int cache();          /* CACHE(3L)          */
int kbdchar();       /* KBDCHAR(3R)       */
void lprintf();     /* PRINTF(3L)       */
int request();      /* RESOURCES(3R)    */
int wait();         /* RESOURCES(3R)    */

main()
{
    /*
    ** Put application in the cache.
    */
    cache((char *)0, 0); /* put this application in the cache */

    lprintf("\n Hello World.");

    /*
    ** Wait for the user to hit a key.
    */
    request(KBD);
    while (kbdchar() == -1) wait(KBD);
}
```

Figure 10-2: Example Program "cache2.c"

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

/* List of routines and associated manual page */
int cache();          /* CACHE(3L)          */
int kbdchar();       /* KBDCHAR(3R)       */
int local();         /* LOCAL(3R)         */
void lprintf();     /* PRINTF(3L)       */
int request();      /* RESOURCES(3R)    */
int wait();         /* RESOURCES(3R)    */

main()
{
    /*
    ** Put application in the cache.
    */
    cache((cache *)0, 0);

    /*
    ** Release the host connection.
    */
    local();

    lprintf("\n Hello World.");

    /*
    ** Wait for the user to hit a key.
    */
    request(KBD);
    while (kbdchar() == -1) wait(KBD);
}
```

Figure 10-3: Example Program "cache3.c"

Example Programs for Application Caching

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

/* List of routines and associated manual page */
int cache();           /* CACHE(3L)           */
int kbdchar();        /* KBDCHAR(3R)         */
int local();          /* LOCAL(3R)           */
void lprintf();       /* PRINTF(3L)          */
int request();        /* RESOURCES(3R)       */
int wait();           /* RESOURCES(3R)       */

main()
{
    /*
    ** Release the host connection.
    */
    local();

    /*
    ** Put application in the cache
    ** as a shared application.
    */
    cache((cache *)0, A_SHARED);

    lprintf("\n Hello World.");

    /*
    ** Wait for the user to hit a key.
    */
    request(KBD);
    while (kbdchar() == -1) wait(KBD);
}
```

Figure 10-4: Example Program "cache4.c"

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

/* List of routines and associated manual page */
int cache();          /* CACHE(3L)          */
int kbdchar();       /* KBDCHAR(3R)       */
int local();         /* LOCAL(3R)         */
void lprintf();     /* PRINTF(3L)       */
int request();      /* RESOURCES(3R)    */
int wait();         /* RESOURCES(3R)    */

main()
{
    /*
    ** Release the host connection.
    */
    local();

    /*
    ** Make the window not reshappable.
    */
    P->state |= NO_RESHAPE;

    /*
    ** Put application in the cache
    ** as a shared application.
    */
    cache((char *)0, A_SHARED);

    lprintf("\n Hello World.");

    /*
    ** Wait for the user to hit a key.
    */
    request(KBD);
    while (kbdchar() == -1) wait(KBD);
}
```

Figure 10-5: Example Program "cache5.c"

Example Programs for Application Caching

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

/* List of routines and associated manual page */
int cache();          /* CACHE(3L)          */
int kbdchar();       /* KBDCHAR(3R)       */
int local();         /* LOCAL(3R)         */
void lprintf();     /* PRINTF(3L)       */
int request();      /* RESOURCES(3R)    */
int wait();         /* RESOURCES(3R)    */

/* Local routines in this file */
Point WindowSize();

main()
{
    /*
    ** Release the host connection.
    */
    local();

    /*
    ** Initialize P->ctob.
    */
    P->ctob = WindowSize;

    /*
    ** Put application in the cache
    ** as a shared application.
    */
    cache((char *)0, A_SHARED);

    lprintf("\n Hello World.");

    /*
    ** Wait for the user to hit a key.
    */
    request(KBD);
    while (kbdchar() == -1) wait(KBD);
}
```

Figure 10-6: Example Program "cache6.c" (Sheet 1 of 2)

```
Point  
WindowSize()  
{  
    Point p;  
    p.x = 250;  
    p.y = 250;  
    return(p);  
}
```

Figure 10-6: Example Program "cache6.c" (Sheet 2 of 2)

Example Programs for Application Caching

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

/* List of routines and associated manual page */
int cache();           /* CACHE(3L)           */
int kbdchar();        /* KBDCHAR(3R)        */
int local();          /* LOCAL(3R)          */
void lprintf();       /* PRINTF(3L)        */
int request();        /* RESOURCES(3R)     */
int wait();           /* RESOURCES(3R)     */

/* Local routines in this file */
Point WindowSize();

main()
{
    /*
    ** Release the host connection.
    */
    local();

    /*
    ** Make window not reshappable.
    */
    P->state != NO_RESHAPE;

    /*
    ** Initialize ctob.
    */
    P->ctob = WindowSize;

    /*
    ** Put application in the cache
    ** as a shared application.
    */
    cache((char *)0, A_SHARED);

    lprintf("\n Hello World.");
}
```

Figure 10-7: Example Program "cache7.c" (Sheet 1 of 2)

```
    /*
    ** Wait for the user to hit a key.
    */
    request(KBD);
    while (kbdchar() == -1) wait(KBD);
}

Point
WindowSize()
{
    Point p;
    p.x = 250;
    p.y = 250;
    return(p);
}
```

Figure 10-7: Example Program "cache7.c" (Sheet 2 of 2)

Example Programs for Application Caching

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

/* List of routines and associated manual page */
int cache();          /* CACHE(3L)          */
int kbdchar();       /* KBDCHAR(3R)       */
int local();         /* LOCAL(3R)         */
void lprintf();     /* PRINTF(3L)       */
int request();      /* RESOURCES(3R)    */
int wait();         /* RESOURCES(3R)    */

/*
** temp1 is an initialized external variable
** that is placed in the data section.
*/
int temp1 = 1;

main()
{
    /*
    ** temp2 is an initialized static variable
    ** that is placed in the data section.
    */
    static temp2 = 2;

    /*
    ** Release the host connection.
    */
    local();

    /*
    ** Put application in the cache and give the
    ** instruction not to reinitialize the data
    ** section to its original contents on subsequent
    ** invocations.
    */
    cache((char *)0, A_DATA);

    lprintf("\n temp1 = %d, temp2 = %d", temp1, temp2);
    temp1++;
    temp2++;
    lprintf("\n temp1 = %d, temp2 = %d", temp1, temp2);
}
```

Figure 10-8: Example Program "cache8.c" (Sheet 1 of 2)

```
/*  
** Wait for the user to hit a key.  
*/  
request(KBD);  
while(kbdchar() == -1) wait(KBD);  
}
```

Figure 10-8: Example Program "cache8.c" (Sheet 2 of 2)

Example Programs for Application Caching

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

/* List of routines and associated manual page */
int cache();           /* CACHE(3L)           */
int kbdchar();        /* KBDCHAR(3R)         */
int local();          /* LOCAL(3R)           */
void lprintf();       /* PRINTF(3L)          */
int request();        /* RESOURCES(3R)       */
int wait();           /* RESOURCES(3R)       */

/*
** temp1 is an uninitialized global variable
** and is placed in the bss section.
*/
int temp1;

main()
{
    /*
    ** temp2 is an uninitialized static variable
    ** and is placed in the bss section.
    */
    static temp2;

    /*
    ** Release the host connection.
    */
    local();

    /*
    ** Put application in the cache
    ** as a shared application.
    */
    cache((char *)0, 0);
}
```

Figure 10-9: Example Program "cache9.c" (Sheet 1 of 2)

```
lprintf("\n temp1 = %d, temp2 = %d", temp1, temp2);
temp1++;
temp2++;
lprintf("\n temp1 = %d, temp2 = %d", temp1, temp2);

/*
** Wait for the user to hit a key.
*/
request(KBD);
while (kbdchar() == -1) wait(KBD);
}
```

Figure 10-9: Example Program "cache9.c" (Sheet 2 of 2)

Example Programs for Application Caching

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

/* List of routines and associated manual page */
int cache();           /* CACHE(3L)           */
int kbdchar();        /* KBDCHAR(3R)        */
int local();          /* LOCAL(3R)          */
void lprintf();       /* PRINTF(3L)         */
int request();        /* RESOURCES(3R)      */
int wait();           /* RESOURCES(3R)      */

/*
** temp1 is an uninitialized global variable
** and is placed in the bss section.
*/
int temp1;

main()
{
    /*
    ** temp2 is an uninitialized static variable
    ** and is placed in the bss section.
    */
    static temp2;

    /*
    ** Release the host connection.
    */
    local();

    /*
    ** Put application in the cache and do not
    ** initialize the bss section to zero on
    ** subsequent invocations. Note that the
    ** bss section is initialized to zero the
    ** first time the application runs.
    */
    cache((char *)0, A_BSS);
}
```

Figure 10-10: Example Program "cache10.c" (Sheet 1 of 2)

```
lprintf("\n temp1 = %d, temp2 = %d", temp1, temp2);
temp1++;
temp2++;
lprintf("\n temp1 = %d, temp2 = %d", temp1, temp2);

/*
** Wait for the user to hit a key.
*/
request(KBD);
while (kbdchar() == -1) wait(KBD);
}
```

Figure 10-10: Example Program "cache10.c" (Sheet 2 of 2)

Example Programs for Application Caching

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

/* List of routines and associated manual page */
int cache();          /* CACHE(3L)          */
int kbdchar();       /* KBDCHAR(3R)        */
int local();         /* LOCAL(3R)          */
void lprintf();     /* PRINTF(3L)         */
int request();      /* RESOURCES(3R)      */
int wait();         /* RESOURCES(3R)      */

typedef struct Globals {
    int global1;
    int global2;
} Globals;

main()
{
    Globals myglobals;

    /*
    ** Release the host connection.
    */
    local();

    /*
    ** Put application in the cache
    ** as a shared application.
    */
    cache((char *)0, A_SHARED);

    myglobals.global1 = 0;
    myglobals.global2 = 0;

    lprintf("\n global1 = %d, global2 = %d",
            myglobals.global1, myglobals.global2);
    myglobals.global1++;
    myglobals.global2++;
    lprintf("\n global1 = %d, global2 = %d",
            myglobals.global1, myglobals.global2);
}
```

Figure 10-11: Example Program "cache11.c" (Sheet 1 of 2)

```
/*
** Wait for the user to hit a key.
*/
request(KBD);
while (kbdchar() == -1) wait(KBD);
}
```

Figure 10-11: Example Program "cache11.c" (Sheet 2 of 2)

Example Programs for Application Caching

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

/* List of routines and associated manual page */
int cache();           /* CACHE(3L)           */
int kbdchar();        /* KBDCHAR(3R)        */
int local();          /* LOCAL(3R)           */
void lprintf();       /* PRINTF(3L)         */
int request();        /* RESOURCES(3R)      */
int wait();           /* RESOURCES(3R)      */

/* local routines */
void print();

typedef struct Globals {
    int global1;
    int global2;
} Globals;

main()
{
    Globals myglobals;

    /*
     ** Release the host connection.
     */
    local();

    /*
     ** Put application in the cache
     ** as a shared application.
     */
    cache((char *)0, A_SHARED);

    myglobals.global1 = 0;
    myglobals.global2 = 0;

    print(&myglobals);
    myglobals.global1++;
    myglobals.global2++;
    print(&myglobals);
}
```

Figure 10-12: Example Program "cache12.c" (Sheet 1 of 2)

```
    /*
    ** Wait for the user to hit a key.
    */
    request(KBD);
    while (kbdchar() == -1) wait(KBD);
}

void
print(globals)
Globals *globals;
{
    lprintf("\n global1 = %d, global2 = %d",
           globals->global1, globals->global2);
}
```

Figure 10-12: Example Program "cache12.c" (Sheet 2 of 2)

Example Programs for Application Caching

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

/* List of routines and associated manual page */
char *alloc();          /* ALLOC(3R)          */
int cache();           /* CACHE(3L)         */
int kbdchar();        /* KBDCHAR(3R)       */
int local();          /* LOCAL(3R)         */
void lprintf();       /* PRINTF(3L)        */
int request();        /* RESOURCES(3R)     */
int wait();           /* RESOURCES(3R)     */

/* local routines */
void print();

typedef struct Globals {
    int global1;
    int global2;
} Globals;

main()
{
    Globals *myglobals;

    /*
    ** Release the host connection.
    */
    local();

    /*
    ** Put application in the cache
    ** as a shared application.
    */
    cache((char *)0, A_SHARED);

    myglobals = (Globals *)alloc(sizeof(Globals));

    print(myglobals);
    myglobals->global1++;
    myglobals->global2++;
    print(myglobals);
}
```

Figure 10-13: Example Program "cache13.c" (Sheet 1 of 2)

```
    /*
    ** Wait for the user to hit a key.
    */
    request(KBD);
    while (kbdchar() == -1) wait(KBD);
}

void
print(globals)
Globals *globals;
{
    lprintf("\n global1 = %d, global2 = %d",
           globals->global1, globals->global2);
}
```

Figure 10-13: Example Program "cache13.c" (Sheet 2 of 2)

Example Programs for Application Caching

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

/* Library routines and associated manual page. */
char *alloc();          /* ALLOC(3R)          */
int cache();           /* CACHE(3L)         */
int kbdchar();        /* KBDCHAR(3R)       */
int local();          /* LOCAL(3R)         */
void lprintf();       /* PRINTF(3L)        */
int request();        /* RESOURCES(3R)     */
int wait();           /* RESOURCES(3R)     */

/* local routines */
void print();

typedef struct Globals {
    int global1;
    int global2;
} Globals;

main()
{
    /*
    ** Release the host connection.
    */
    local();

    /*
    ** Put application in the cache
    ** as a shared application.
    */
    cache((char *)0, A_SHARED);

    P->appl = (long)alloc(sizeof(Globals));

    print();
    (((struct Globals *) (P->appl))->global1)++;
    (((struct Globals *) (P->appl))->global2)++;
    print();
}
```

Figure 10-14: Example Program "cache14.c" (Sheet 1 of 2)

```
    /*
    ** Wait for the user to hit a key.
    */
    request(KBD);
    while (kbdchar() == -1) wait(KBD);
}

void
print()
{
    lprintf("\n global1 = %d, global2 = %d",
            (((struct Globals *) (P->appl))->global1),
            (((struct Globals *) (P->appl))->global2));
}
```

Figure 10-14: Example Program "cache14.c" (Sheet 2 of 2)

Example Programs for Application Caching

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

/* Library routines and associated manual page. */
char *alloc();          /* ALLOC(3R)          */
int cache();           /* CACHE(3L)         */
int kbdchar();        /* KBDCHAR(3R)       */
int local();          /* LOCAL(3R)         */
void lprintf();       /* PRINTF(3L)        */
int request();        /* RESOURCES(3R)     */
int wait();           /* RESOURCES(3R)     */

/* local routines */
void print();

typedef struct Globals {
    int _global1;
    int _global2;
} Globals;

#define global1      (((struct Globals *) (P->appl))->_global1)
#define global2      (((struct Globals *) (P->appl))->_global2)

main()
{
    /*
    ** Release the host connection.
    */
    local();

    /*
    ** Put application in the cache
    ** as a shared application.
    */
    cache((char *)0, A_SHARED);

    P->appl = (long)alloc(sizeof(Globals));
}
```

Figure 10-15: Example Program "cache15.c" (Sheet 1 of 2)

```
    print();
    global1++;
    global2++;
    print();

    /*
    ** Wait for the user to hit a key.
    */
    request(KBD);
    while (kbdchar() == -1) wait(KBD);
}

void
print()
{
    lprintf("\n global1 = %d, global2 = %d", global1, global2);
}
```

Figure 10-15: Example Program "cache15.c" (Sheet 2 of 2)

Example Programs for Application Caching

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

/* List of routines and associated manual page */
char *alloc();          /* ALLOC(3R)          */
int cache();           /* CACHE(3L)         */
int kbdchar();         /* KBDCHAR(3R)       */
int local();           /* LOCAL(3R)         */
void lprintf();        /* PRINTF(3L)        */
int request();         /* RESOURCES(3R)     */
int wait();            /* RESOURCES(3R)     */

/* local routines */
void print();
Point SetSize();

typedef struct Globals {
    int _global1;
    int _global2;
    int _width;
    int _height;
} Globals;

#define global1      (((struct Globals *) (P->appl))->_global1)
#define global2      (((struct Globals *) (P->appl))->_global2)
#define width        (((struct Globals *) (P->appl))->_width)
#define height       (((struct Globals *) (P->appl))->_height)

main()
{
    /*
    ** Release the host connection.
    */
    local();
    /*
    ** Initialize ctob.
    */
    P->ctob = SetSize;

    P->appl = (long)alloc(sizeof(Globals));
    width = 325;
    height = 100;
}
```

Figure 10-16: Example Program "cache16.c" (Sheet 1 of 2)

```
/*
** Put application in the cache
** as a shared application.
*/
cache((char *)0, A_SHARED);

print();
global1++;
global2++;
print();

/*
** Put application in the cache
** as a shared application.
*/
request(KBD);
while (kbdchar() == -1) wait(KBD);
}

void
print()
{
    lprintf("\n global1 = %d, global2 = %d", global1, global2);
}

Point
SetSize(x, y, p)
int x, y;
struct Proc *p;
{
    Point pt;

    pt.x = (((struct Globals *) (p->appl))->_width);
    pt.y = (((struct Globals *) (p->appl))->_height);
    return(pt);
}
```

Figure 10-16: Example Program "cache16.c" (Sheet 2 of 2)

Example Programs for Application Caching

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

/*
** Constant definitions.
*/
#define MAX_TARGET      20          /* maximum number of targets */
#define MSG_TYPE        (long)1    /* message type on global queue */
#define MSG_SIZE        8          /* size of my message */
#define GLB_Q_KEY       0x44454d4f /* Key for Global Queue is DEMO */
#define SS              3          /* size of a shot */
#define FAILURE         -1         /* message get failure */

/*
** Type Definitions.
*/
typedef struct MyMsgBuf {
    long    mtype;
    Point   WindowCenter;
    long    LocalMsgQid;
} MyMsgBuf;

typedef struct Shot {
    Point   ShotCenter;
    Point   Velocity;
} Shot;

/*
** Global Variables.
*/
typedef struct Globals {
    Point   _Target[MAX_TARGET];
    int     _NumOfTargets;
}Globals;

#define Target          (((struct Globals *) (P->appl))->_Target)
#define NumOfTargets    (((struct Globals *) (P->appl))->_NumOfTargets)
```

Figure 10-17: Example Program "cache17.c" (Sheet 1 of 8)

```
/*Library routines and associated manual pages. */
Point add();          /* PTARITH(3R) */
char *alloc();       /* ALLOC(3R) */
Point div();         /* PTARITH(3R) */
int kbdchar();      /* KBDCHAR(3R) */
int local();        /* LOCAL(3R) */
long msgget();      /* MSGGET(3L) */
int msgctl();       /* MSGCTL(3L) */
int msgsnd();       /* MSGOP(3L) */
int msgrcv();       /* MSGOP(3L) */
int request();      /* RESOURCES(3R) */
void sleep();       /* SLEEP(3R) */
Point sub();        /* PTARITH(3R) */

/* local routines */
void AddToList();
void DrawShots();
void Failure();
Point GetWindowCenter();
void Shoot();
Point WindowSize();

main()
{
    MyMsgBuf MyMsg, TempMsg;
    long GlbMsgQid;
    msqid_ds LocalMsgQ;
    int i;

    /*
     ** This application does not need a host connection.
     */
    local();

    /*
     ** Dynamically allocate global variables.
     */
    P->appl = (long)alloc(sizeof(Globals));
}
```

Figure 10-17: Example Program "cache17.c" (Sheet 2 of 8)

Example Programs for Application Caching

```
/*
** Init ctob function.
*/
P->ctob = WindowSize;

/*
** Cache as a local shared text application,
** and advertise the application with the name
** "Messages".
*/
cache("Messages", A_SHARED);

/*
** Request the use of the Keyboard.
*/
request(KBD);

if((GlbMsgQid = msgget(GLB_Q_KEY, IPC_CREAT)) == FAILURE)
/*
** Could not open global message queue.
*/
Failure("Can't get global message queue");

MyMsg.LocalMsgQid = msgget((long)IPC_PRIVATE, IPC_CREAT|NO_SAVE);
if(MyMsg.LocalMsgQid == FAILURE)
/*
** Could not open my local message queue.
*/
Failure("Can't create local message queue");

MyMsg.mtype = MSG_TYPE;
MyMsg.WindowCenter = GetWindowCenter();
if(msgsnd(GlbMsgQid, &MyMsg, MSG_SIZE, IPC_NOWAIT))
/*
** Could not put my message into the global message queue.
*/
Failure("Can't send my first message");
```

Figure 10-17: Example Program "cache17.c" (Sheet 3 of 8)

```
/*
** Main loop of program.
*/
while (kbdchar() != 'q') {
    if (msgctl(MyMsg.LocalMsgQid, IPC_STAT, &LocalMsgQ))
        /*
        ** Someone has removed my local message queue.
        ** They must want me dead.
        */
        Failure("Someone wants me dead");

    if (msgctl(GlbMsgQid, IPC_STAT, &LocalMsgQ))
        /*
        ** Someone has removed the global message queue.
        */
        Failure("global queue is gone");

    /*
    ** Get number of messages currently in the
    ** global message queue.
    */
    i = LocalMsgQ.msg_qnum;
    NumOfTargets = 0;

    /*
    ** If my window was reshaped, erase the window and
    ** recalculate the center of the window.
    */
    if(P->state & RESHAPED) {
        if(P->state & MOVED)
            rectf(&display, Drect, F_CLR);
        P->state &= ~RESHAPED;
        MyMsg.WindowCenter = GetWindowCenter();
    }

    /*
    ** Read all the messages off the global message queue.
    ** If the application that sent the message still exists,
    ** then add that process to the list of targets and put
    ** that applications message back on the global queue.
    ** Otherwise just throw the message away and read the
    ** next message from the queue.
    */
}
```

Figure 10-17: Example Program "cache17.c" (Sheet 4 of 8)

Example Programs for Application Caching

```
while(i-- &&
    msgrcv(GlbMsgQid, &TempMsg,
        MSG_SIZE, MSG_TYPE, IPC_NOWAITMSG_NOERROR) != -1) {
    if(msgctl(TempMsg.LocalMsgQid, IPC_STAT, &LocalMsgQ) == 0) {
        /*
        ** Application still exists.
        */
        if(TempMsg.LocalMsgQid != MyMsg.LocalMsgQid)
            /*
            ** If it is not my own message,
            ** then add it to the list of targets.
            */
            AddToList(TempMsg.WindowCenter);
        else
            /*
            ** I just read my own message.
            ** Make sure my widow center is up
            ** to date.
            */
            TempMsg.WindowCenter = MyMsg.WindowCenter;

        /*
        ** Put the message back on the global queue.
        */
        if (msgsnd(GlbMsgQid, &TempMsg, MSG_SIZE, IPC_NOWAIT))
            /*
            ** Oh Oh! The message send has fail.
            ** Remove the queue that failed.
            */
            msgctl(TempMsg.LocalMsgQid, IPC_RMID, &LocalMsgQ);
    }
}

/*
** If i is greater than or equal to zero, then for some reason
** we were not able to read all the messages from the global
** message queue. Better quit.
*/
if (i >= 0)
    Failure("I can't read from the global queue");
```

Figure 10-17: Example Program "cache17.c" (Sheet 5 of 8)

Example Programs for Application Caching

```
    /*
    ** Shoot bullets at the other windows running this application.
    */
    Shoot();

    /*
    ** In case the number of targets was zero, switch out
    ** here to let other applications run
    */
    sleep(1);
}

void
AddToList(p)
Point p;
/*
** Add a target to the list.
*/
{
    if (NumOfTargets == MAX_TARGET)
        /*
        ** Over the maximum target limit.
        */
        return;

    /*
    ** Add target to the list
    */
    Target[NumOfTargets++] = p;
}

void
Shoot()
{
```

Figure 10-17: Example Program "cache17.c" (Sheet 6 of 8)

Example Programs for Application Caching

```
int i, j;
Shot shot[MAX_TARGET];

for (i=0; i<NumOfTargets; i++) {
    shot[i].ShotCenter = GetWindowCenter();
    shot[i].Velocity = div(sub(Target[i], shot[i].ShotCenter), 50);
}

DrawShots(shot);
for (j=0; j<50; j++) {
    sleep(5);
    if (P->state & RESHAPED)
        return;

    /*
    ** Erase the old shots.
    */
    DrawShots(shot);

    /*
    ** Update the center position of each shot.
    */
    for (i=0; i<NumOfTargets; i++)
        shot[i].ShotCenter = add(shot[i].ShotCenter, shot[i].Velocity);

    /*
    ** Draw the new shots.
    */
    DrawShots(shot);
}

/*
** Erase all shots.
*/
DrawShots(shot);
}

void
DrawShots(shot)
Shot *shot;
```

Figure 10-17: Example Program "cache17.c" (Sheet 7 of 8)

```
{
    int i;

    for(i=0; i<NumOfTargets; i++)
        /*
        ** Draw the shot in the display Bitmap.
        */
        rectf(&display,
            Rpt(sub(shot[i].ShotCenter, Pt(SS,SS)),
                add(shot[i].ShotCenter, Pt(SS,SS))),
            F_XOR);
}

void
Failure(string)
char *string;
{
    /*
    ** Print error message and wait for user to hit q.
    */
    jstring(string);
    request(KBD);
    while (kbdchar() != 'q') wait(KBD);
    exit();
}

Point
GetWindowCenter()
{
    /*
    ** Calculate the center Point of the window
    */
    return add(div(sub(Direct.corner, Direct.origin), 2), Direct.origin);
}

Point
WindowSize()
{
    Point p;
    p.x = 200;
    p.y = 200;
    return(p);
}
```

Figure 10-17: Example Program "cache17.c" (Sheet 8 of 8)

Chapter 11: Redefining Keyboard Operations

Introduction	11-1
Redefining Key Clusters	11-2
Function Keys	11-2
Example of Redefining Function Keys (NOPFEXPAND) - "kbd1.c"	11-3
Cursor Keys	11-3
Numeric Key Pad	11-4
Scroll Lock Key	11-5
Redefining the Entire Keyboard	11-6
630 MTG Keyboard Operation	11-6
NOTRANSULATE Protocol	11-6
Keystrokes	11-7
Keyboard Identification	11-7
Process Switch Notification	11-8
Demonstrations of Keyboard Redefinition	
Modes	11-9
Keyboard Modes and Process Switch Notification - "kbd2.c"	11-9
Graphical Demonstration of the NOTRANSULATE Mode - "kbd3.c"	11-9
Example Programs	11-11
Keyboard Transmittal Codes and Keyboard Layout	11-21

Introduction

When an application requests the keyboard resource in the following manner:

```
request (KBD) ;
```

the results of each subsequent keystroke are placed on the application's keyboard queue. Data is then read from the keyboard queue one character at a time using the routine **kbdchar**.

Many times it is convenient to redefine the functions of certain groups of keys in order to simplify processing. For example, you may want to redefine the cluster of cursor keys to eliminate processing the corresponding escape sequences. At other times you may want to redefine the entire keyboard to suit the special needs of your application. These two features, redefining groups of keys and redefining the entire keyboard, are discussed in this chapter.

Redefining Key Clusters

The following four key clusters can be redefined by applications:

1. Function keys
2. Cursor keys
3. Numeric key pad
4. Scroll Lock key.

Function Keys

An application can suppress the normal definition of the programmable and static function keys by requesting the keyboard resource and setting the NOPFEXPAND bit in the application's state variable as shown below:

```
request(KBD);  
P->state |= NOPFEXPAND;
```

This will cause the following predefined constants, defined in the include file *\$DMD/include/keycodes.h*, to be placed in the keyboard queue when a function key is depressed.

Function Key	Constant Definition	8-bit code
F1	FUNC1KEY	0x80
F2	FUNC2KEY	0x81
F3	FUNC3KEY	0x82
F4	FUNC4KEY	0x83
F5	FUNC5KEY	0x84
F6	FUNC6KEY	0x85
F7	FUNC7KEY	0x86
F8	FUNC8KEY	0x87
F9	FUNC9KEY	0x88
F10	FUNC10KEY	0x89
F11	FUNC11KEY	0x8a
F12	FUNC12KEY	0x8b
F13	FUNC13KEY	0x8c
F14	FUNC14KEY	0x8d

Each of the 8-bit constants defined in the previous table has the most significant bit set to "1". This allows your application to differentiate the function key constants from other characters received from the keyboard.

Example of Redefining Function Keys (NOPFEXPAND) - "kbd1.c"

`kbd1.c` demonstrates how the normal definition of the programmable and static function keys can be redefined. The source code for `kbd1.c` is provided in *\$DMD/examples/Keyboard*. A printout of `kbd1.c` is given in Figure 11-1 at the end of this chapter. Compile and download this program using the following commands:

```
dmdcc -o kbd1 kbd1.c
dmdld kbd1
```

Cursor Keys

The 98-key keyboard contains a cluster of cursor keys which are the four "arrow" keys and the "Home" key.

Each cursor key is assigned a predefined escape sequence. (See the *630 MTG Terminal User's Guide* for a list of the cursor key escape sequences.) To avoid the overhead of processing escape sequences, an application that has requested the keyboard can set the `NOCURSEXPAND` bit in the process state variable. When `P->state&NOCURSEXPAND` is true, the following predefined constants, defined in the include file *\$DMD/include/keycodes.h*, will be placed on the application's keyboard queue for each cursor key:

CURSOR KEY	Constant Definition	8-bit Code
up arrow	UP_ARROW	0xe0
down arrow	DOWN_ARROW	0xe1
right arrow	RIGHT_ARROW	0xe2
left arrow	LEFT_ARROW	0xe3
home	HOME_KEY	0xe4

Each of the 8-bit constants defined in the previous table has the three most significant bits set to "1". This allows your application to differentiate between the cursor key constants and other characters received from the keyboard.

Numeric Key Pad

The numeric key pad on the keyboard can be redefined by requesting the keyboard resource and setting the NOPADEXPAND bit in the process state variable:

```
request(KBD);  
P->state |= NOPADEXPAND;
```

When `P->state&NOPADEXPAND` is true, the predefined constants received in response to a key pad depression are shown below:

Pad Key	Constant Definition	8-bit Code
Enter	PAD_ENTER	0xc0
Equals	PAD_EQUALS	0xc1
Asterisk	PAD_ASTERISK	0xc2
Slash	PAD_SLASH	0xc3
Plus	PAD_PLUS	0xc4
Seven	PAD_7	0xc5
Eight	PAD_8	0xc6
Nine	PAD_9	0xc7
Minus	PAD_MINUS	0xc8
Four	PAD_4	0xc9
Five	PAD_5	0xca
Six	PAD_6	0xcb
Comma	PAD_COMA	0xcc
One	PAD_1	0xcd
Two	PAD_2	0xce
Three	PAD_3	0xcf
Zero	PAD_0	0xd0
Dot	PAD_DOT	0xd1

Each of the 8-bit constants defined in the previous table have the two most significant bits set to "1". This allows your application to differentiate between the key pad constants and other characters received from the keyboard.

Scroll Lock Key

The 630 MTG keyboard has a Scroll Lock key and an associated status LED. This key is a "dead" key (no special processing) unless it is requested and processed by an application. The function of the Scroll Lock key is controlled by the application that has ownership of the keyboard, which is the current window. For example, the terminal emulator **Windowproc** stops reading characters received from the host when a user hits the Scroll Lock key. This has the effect of freezing the display until the user depresses the Scroll Lock key again.

To define the function of the Scroll Lock key, your application must request it by setting a bit in the process state variable:

```
P->state |= SCRLOCKREQ;
```

Your application can then determine if the Scroll Lock key has been depressed by checking the SCR_LOCK bit in the state variable:

```
if (P->state & SCR_LOCK) {  
    process_scroll_lock();  
}
```

The SCR_LOCK bit in the process state variable is set whenever the Scroll Lock key is depressed. Therefore, the condition **P->state&SCR_LOCK** will remain true until the user depresses the Scroll Lock key a second time. When the Scroll Lock key is requested as shown above, the Scroll Lock LED will be properly updated by the 630 MTG keyboard process. Also, if your application uses the "label bar," the Scroll Lock icon will turn on and off appropriately.

Redefining the Entire Keyboard

Your application can turn off the normal translation performed by the 630 MTG keyboard process by setting the NOTTRANSLATE bit in the process state variable.

```
P->state |= NOTTRANSLATE;
```

Setting the NOTTRANSLATE bit enables your application to totally redefine the keyboard. In order to properly use this capability, you need to be familiar with the 630 MTG keyboard operation and the NOTTRANSLATE protocol between the keyboard and a requesting application.

630 MTG Keyboard Operation

The 630 MTG keyboard is a transparent keyboard. The fundamental characteristic of a transparent keyboard is that each key transmits a unique 8-bit code for a downstroke and an upstroke. The code sent for each downstroke and upstroke is equivalent except that the most significant bit is set to one for a "downstroke." For example, the "raw code" for the key corresponding to the letter 'A' has the hexadecimal value 0x5c. For the downstroke, the value sent by the keyboard is 0xdc; for an upstroke, it is 0x5c. See the table of transmitted codes for the 98-key keyboard at the end of this chapter (Figure 11-4).

NOTTRANSLATE Protocol

When the NOTTRANSLATE bit is set in the process state variable, three different data types can be placed on the keyboard queue:

- Keystroke
- Keyboard identification
- Process switch notification.

Each data type is sent in a two-byte packet: the first byte identifies the data type, and the second byte contains the data. The following sections describe these different data types.

Keystrokes

Keystrokes are identified by the predefined constant "KEYSTROKE". For example, when the 'A' key is depressed, the following two bytes will be placed in the keyboard input queue:

```
<KEYSTROKE><0xdc>
```

When the 'A' key is released, the two bytes

```
<KEYSTROKE><0x5c>
```

will be placed in the queue. Each keystroke, whether it is a downstroke or an upstroke, is preceded by the constant "KEYSTROKE".

Keyboard Identification

The routine `reqkbdID` is used to request the "keyboard identification sequence". The keyboard identification sequence has the following format:

```
[keyboard ID] [keys currently depressed] [keyboard ID]
```

The first segment of the keyboard identification sequence [keyboard ID] consists of two bytes. The first byte is a predefined constant called "BEFOREID" that identifies the next byte as a "keyboard ID", and the second byte is the "keyboard ID" itself as shown in the following format:

```
<BEFOREID><KEY98ID>
```

The second segment of the keyboard identification sequence [keys currently depressed] is a report of all the keys that are currently depressed. The data received in this segment will look like multiple key downstrokes. For example, if the keys 'A' and 'CTRL' are currently depressed, the data placed on the keyboard queue will have the following format:

```
<KEYSTROKE><0xdc><KEYSTROKE><0xc4>
```

The third segment of the keyboard identification sequence [keyboard ID] terminates the entire sequence. Therefore, if the 'A' and 'CTRL' keys are depressed, the entire "keyboard identification sequence" would be:

```
<BEFOREID><KEY98ID>  
<KEYSTROKE><0xdc><KEYSTROKE><0xc4>  
<BEFOREID><KEY98ID>
```

Redefining the Entire Keyboard

If no keys are depressed when **reqkbdID** is called, the keyboard identification sequence will consist of two back-to-back "keyboard IDs" as shown in the following format:

```
<BEFOREID><KEY98ID><BEFOREID><KEY98ID>
```

When your application is using the NOTRANSLATE mode and is made the current window, **reqkbdID** is called by the 630 MTG operating system to notify your application of the current status of the keyboard. See the manual page **KEYBOARD(3R)** for more details on **reqkbdID**.

Process Switch Notification

The "process switch notification" is used to notify an application, which is running in the current window in the NOTRANSLATE mode, that another window has been made current and owns the keyboard. The two-byte packet received has the following format:

```
<SWITCHCHAR><SWITCHCHAR>
```

Note: The constants **KEYSTROKE**, **BEFOREID**, **KEY98ID**, and **SWITCHCHAR** are defined in the include file *\$DMD/example/Keyboard/keycodes.h*.

Demonstrations of Keyboard Redefinition Modes

Keyboard Modes and Process Switch Notification - "kbd2.c"

kbd2.c is a menu-driven application that demonstrates the different modes: NOPFEXPAND, NOCURSEXPAND, NOPADEXPAND, and NOTRANSLATE. The source code for **kbd2.c** is provided in *\$DMD/examples/Keyboard*, and a printout is given in Figure 11-2.

After **kbd2.c** has been compiled and downloaded, depressing mouse button 2 brings up the application's menu. The application will set the mode specified by the menu selection and will print out the hexadecimal values of data received from the keyboard. This allows you to view how each mode described in this chapter operates. The current mode(s) is marked by a check mark in the menu.

kbd2.c can also be used to demonstrate the process switch notification protocol. Select the **No Translate** mode from the menu on mouse button 2, and then make another window current. Notice the process switch notification. Now, make the **kbd2.c** window current again. This causes a keyboard identification sequence to be sent that consists of two back-to-back "keyboard IDs". Make another window current and depress and hold two or three keys. With the keys depressed, make the **kbd2.c** window current and notice the change in the keyboard identification sequence. This time **reqkbdID** reports not only the "keyboard ID" but also the keys that are currently depressed.

Graphical Demonstration of the NOTRANSLATE Mode - "kbd3.c"

kbd3.c is a graphical demonstration of the NOTRANSLATE mode. When the example starts running, it draws a picture of the keys on the keyboard. As you depress keys on the keyboard, the corresponding key in the picture will be highlighted. The source code for **kbd3.c** is provided in *\$DMD/examples/Keyboard* and a printout is shown in Figure 11-3.

Demonstrations of Keyboard Redefinition Modes

The important data structures for this example are the two arrays **Positions** and **Keys** found in the include files `$DMD/examples/Keyboard/positions.h` and `$DMD/examples/Keyboard/keys.h`, respectively. **Positions** is an array of "key positions" indexed by the "raw code" received from the keyboard, with the most significant bit set to zero. For example, the 'A' key sends the code 0xdc when depressed. Masking the most significant bit gives an index of 0x5c, which corresponds to key position 66. (See the table of transmittal codes in Figure 11-4 for "raw code" and "key position" data.)

The "key position" obtained from the array **Positions** is used to index the array of "Key" structures, **Keys**, which contains the current status of the key, the relative location of the key on the screen, and rectangle type used to draw the key. See the array **KeyRects** in the include file:

`$DMD/examples/keyboard/keyrects.h`

for the different rectangle types used. Figures 11-5 and 11-6 illustrate the layout of the 98-key keyboard and the associated key positions, respectively.

Example Programs

The source code for the example programs on redefining the keyboard is included in this section (Figures 11-1, 11-2, and 11-3).

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

/*Library routines and associated manual pages. */
void exit();           /* EXIT(3R)           */
int kbdchar();         /* KBDCHAR(3R)       */
void lprintf();        /* PRINTF(3L)        */
void rectf();          /* RECTF(3R)         */
int request();         /* RESOURCES(3R)     */
void texture();        /* TEXTURE(3R)       */
int wait();            /* RESOURCE(3R)      */

extern Texture16 T_lightgrey;
extern Texture16 T_darkgrey;

main()
{
    unsigned char c;

    request(KBD);

    /*
    ** Set no pf key expansion.
    */
    P->state |= NOPFEXPAND;

    lprintf("\n Hit any PF Key.");
    lprintf("\n Use 'q' to quit");
}
```

Figure 11-1: Example Program "kbd1.c" (Sheet 1 of 3)

Example Programs

```
for(;;) {
    wait(KBD);
    c = (unsigned char)kbdchar();
    switch(c) {
        /*
        ** Display a different texture for each
        ** function key hit.
        */
        case FUNC1KEY:
            texture(&display, Drect, &T_grey, F_STORE);
            break;
        case FUNC2KEY:
            texture(&display, Drect, &T_lightgrey, F_STORE);
            break;
        case FUNC3KEY:
            texture(&display, Drect, &T_darkgrey, F_STORE);
            break;
        case FUNC4KEY:
            texture(&display, Drect, &T_black, F_STORE);
            break;
        case FUNC5KEY:
            texture(&display, Drect, &T_white, F_STORE);
            break;
        case FUNC6KEY:
            texture(&display, Drect, &T_background, F_STORE);
            break;
        case FUNC7KEY:
            texture(&display, Drect, &T_checks, F_STORE);
            break;
        case FUNC8KEY:
            texture(&display, Drect, &C_target, F_STORE);
            break;
        case FUNC9KEY:
            texture(&display, Drect, &C_arrows, F_STORE);
            break;
        case FUNC10KEY:
            texture(&display, Drect, &C_insert, F_STORE);
            break;
    }
}
```

Figure 11-1: Example Program "kbd1.c" (Sheet 2 of 3)

```
    case FUNC11KEY:
        texture(&display, Drect, &C_cup, F_STORE);
        break;
    case FUNC12KEY:
        texture(&display, Drect, &C_deadmouse, F_STORE);
        break;
    case FUNC13KEY:
        texture(&display, Drect, &C_skull, F_STORE);
        break;
    case FUNC14KEY:
        rectf(&display, Drect, F_XOR);
        break;
    case 'q':
        exit();
}
}
```

Figure 11-1: Example Program "kbd1.c" (Sheet 3 of 3)

Example Programs

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

/*Library routines and associated manual pages. */
void exit();           /* EXIT(3R)           */
int kbdchar();        /* KBDCHAR(3R)       */
int local();          /* LOCAL(3R)         */
void lprintf();       /* PRINTF(3L)        */
int request();        /* RESOURCES(3R)     */
void sleep();         /* ALARM(3R)         */
Titem *tmenuhit();   /* TMENUHIT(3R)      */
int wait();           /* RESOURCE(3R)      */

/*Library macros and associated manual pages */
/*int button23()      BUTTONS(3R)      */

#define KBD_NORM      0
#define KBD_NOTRAN    1
#define KBD_NOPFXPAN  2
#define KBD_NOPADXPAN 3
#define KBD_NOCURXPAN 4
#define KBD_EXIT      5

Titem kbditems[] = {
    "Normal",           KBD_NORM,      0, 0, 0, 0, 0, 0, 0,
    "No Translate",     KBD_NOTRAN,    0, 0, 0, 0, 0, 0, 0,
    "No PF Expand",     KBD_NOPFXPAN,  0, 0, 0, 0, 0, 0, 0,
    "No Pad Expand",    KBD_NOPADXPAN, 0, 0, 0, 0, 0, 0, 0,
    "No Cursor Expand", KBD_NOCURXPAN, 0, 0, 0, 0, 0, 0, 0,
    "Exit",             KBD_EXIT,      0, 0, 0, 0, 0, 0, 0,
    (char *)0
};

Tmenu kbdmenu = { kbditems };
```

Figure 11-2: Example Program "kbd2.c" (Sheet 1 of 4)

```
main()
{
    int c;
    Titem *titemptr;
    int kbdstart = 0;

    /*
    ** Release the host connection.
    */

    local();
    /*
    ** Place check mark in menu next to
    ** current mode.
    */
    kbditems[KBD_NORM].icon = &B_checkmark;

    /*
    ** Request keyboard and mouse resources.
    */
    request(KBD | MOUSE);

    /*
    ** Main loop of program.
    */
    while (1) {
        /*
        ** Wait on MOUSE, means wait until I'm current.
        ** Wait on KBD, means wait until I receive a char.
        */
        wait(KBD | MOUSE);
        switch(button23()) {
            case 1: /* button 3 */
                /*
                ** Let control process handle
                ** button 3.
                */
                request(KBD);
                sleep(2);
                request(KBD | MOUSE);
                break;
        }
    }
}
```

Figure 11-2: Example Program "kdb2.c" (Sheet 2 of 4)

Example Programs

```
case 2:          /* button 2 */
    if ((titemptr = tmenuhit(&kbdmenu, 2, 0)) == (Titem *) 0)
        break;
    switch(titemptr->ufield.uval) {
    /*
    ** Normal Keyboard processing.
    */
    case KBD_NORM:
        kbdtitems[KBD_NORM].icon = &B_checkmark;
        kbdtitems[KBD_NOTRAN].icon = 0;
        kbdtitems[KBD_NOPFXPAN].icon = 0;
        kbdtitems[KBD_NOPADXPAN].icon = 0;
        kbdtitems[KBD_NOCURXPAN].icon = 0;
        P->state &= ~(NOPFEXPAND | NOCURSEXPAND |
                     NOPADEXPAND | NOTRANSLATE);
        break;
    /*
    ** Turn of the normal translation of
    ** all keys typed on the keyboard.
    */
    case KBD_NOTRAN:
        kbdtitems[KBD_NOTRAN].icon = &B_checkmark;
        kbdtitems[KBD_NORM].icon = 0;
        kbdtitems[KBD_NOPFXPAN].icon = 0;
        kbdtitems[KBD_NOPADXPAN].icon = 0;
        kbdtitems[KBD_NOCURXPAN].icon = 0;
        P->state |= NOTRANSLATE;
        P->state &= ~(NOPFEXPAND | NOCURSEXPAND |
                     NOPADEXPAND);
        break;
    /*
    ** Turn off pf key expansion.
    */
    case KBD_NOPFXPAN:
        kbdtitems[KBD_NOPFXPAN].icon = &B_checkmark;
        kbdtitems[KBD_NORM].icon = 0;
        kbdtitems[KBD_NOTRAN].icon = 0;
        P->state |= NOPFEXPAND;
        P->state &= NOTRANSLATE;
        break;
```

Figure 11-2: Example Program "kbd2.c" (Sheet 3 of 4)

```

/*
** Turn off expansion of keypad.
*/
case KBD_NOPADXPAN:
    kbditems[KBD_NOPADXPAN].icon = &B_checkmark;
    kbditems[KBD_NORM].icon = 0;
    kbditems[KBD_NOTRAN].icon = 0;
    P->state |= NOPADEXPAND;
    P->state &= ~NOTRANSLATE;
    break;

/*
** Turn of expansion of cursor keys.
*/
case KBD_NOCURXPAN:
    kbditems[KBD_NOCURXPAN].icon = &B_checkmark;
    kbditems[KBD_NORM].icon = 0;
    kbditems[KBD_NOTRAN].icon = 0;
    P->state |= NOCURSEXPAND;
    P->state &= ~NOTRANSLATE;
    break;
case KBD_EXIT:
    exit();
    break;
}
}
/*
** display data received from the keyboard.
*/
while((c = kbdchar()) != -1) {
    kbdstart=1;
    lprintf("\n hex=0x%x", c);
}
if (kbdstart) {
    kbdstart = 0;
    lprintf("\n");
}
}
}

```

Figure 11-2: Example Program "kbd2.c" (Sheet 4 of 4)

Example Programs

```
#include <dmd.h>
#include <keycodes.h>
#include "positions.h"
#include "keys.h"
#include "keyrects.h"

/*Library routines and associated manual pages. */
Point add();           /* PTARITH(3R)      */
void box();           /* BOX(3R)         */
int cache();          /* CACHE(3L)       */
Rectangle inset();    /* INSET(3R)       */
int kbdchar();        /* KBDCHAR(3R)     */
int local();          /* LOCAL(3R)       */
int min();            /* INTEGER(3R)     */
Point mul();          /* PTARITH(3R)     */
Rectangle raddp();    /* RECTARITH(3R)   */
void rectf();         /* RECTF(3R)       */
int request();        /* RESOURCES(3R)   */
int wait();           /* RESOURCES(3R)   */

/* Local routines */
void ClearAllKeys();
void DrawKeyboard();
Rectangle GetKeyRect();
void PaintTheKey();

main()
{
    unsigned char c;

    local();
    cache("Keyboard", 0);

    request(KBD);
    P->state != NOTTRANSLATE;

    DrawKeyboard();
    for(;;) {
        wait(KBD);
        c = (unsigned char)kbdchar();
        switch(c) {
```

Figure 11-3: Example Program "kbd3.c" (Sheet 1 of 3)

```
    case KEYSTROKE:
        c = (unsigned char)kbdchar();
        if (c & DOWNSTROKE)
            PaintTheKey(c & ~DOWNSTROKE, 1);
        else
            PaintTheKey(c, 0);
        break;
    case BEFOREID:
        c = (unsigned char)kbdchar();
        break;
    case SWITCHCHAR:
        c = (unsigned char)kbdchar();
        ClearAllKeys();
    }
}

void
PaintTheKey(RawCode, KeyDown)
unsigned char RawCode;
int KeyDown;
{
    int KeyPosition;
    KeyPosition = Positions[RawCode];
    /*
    ** If the state of the key has changed, then
    ** F_XOR the keys rectangle.
    */
    if (Keys[KeyPosition].depressed != KeyDown) {
        Keys[KeyPosition].depressed = KeyDown;
        rectf(&display, GetKeyRect(KeyPosition), F_XOR);
    }
}

void
ClearAllKeys()
{
    int KeyPosition;
    Rectangle r;
    /*
    ** Clear all the rectangles for each key.
    */
}
```

Figure 11-3: Example Program "kbd3.c" (Sheet 2 of 3)

Example Programs

```
for(KeyPosition=0; KeyPosition<128; KeyPosition++) {
    if((Keys[KeyPosition].RectType != -1) &&
        (Keys[KeyPosition].depressed)) {
        Keys[KeyPosition].depressed = 0;
        rectf(&display, r = GetKeyRect(KeyPosition), F_CLR);
        box(&display, r, F_STORE);
    }
}

void
DrawKeyboard()
{
    int KeyPosition;
    for(KeyPosition=0; KeyPosition<128; KeyPosition++) {
        if(Keys[KeyPosition].RectType != -1) {
            Keys[KeyPosition].depressed = 0;
            box(&display, GetKeyRect(KeyPosition), F_STORE);
        }
    }
}

Rectangle
GetKeyRect(KeyPosition)
{
    Point TmpPoint;
    Rectangle TmpRect;
    int mx, my, mult;

    /*
    ** Return the rectangle for a given key position.
    */
    mx = (Drect.corner.x - Drect.origin.x)/125 + 1;
    my = (Drect.corner.y - Drect.origin.y)/125 + 1;
    mult = min(mx, my);

    TmpRect = KeyRects[Keys[KeyPosition].RectType];
    TmpRect = raddp(TmpRect, Keys[KeyPosition].Location);
    TmpRect.origin = mul(TmpRect.origin, mult);
    TmpRect.corner = mul(TmpRect.corner, mult);
    TmpPoint = add(display.rect.origin, Pt(2*mult, 2*mult));
    TmpRect = raddp(TmpRect, TmpPoint);
    return(TmpRect);
}
```

Figure 11-3: Example Program "kbd3.c" (Sheet 3 of 3)

Keyboard Transmittal Codes and Keyboard Layout

Illustrations of the key transmittal codes, the keyboard layout, and keyboard key positions are included in this section (Figures 11-4, 11-5, and 11-6).

Keyboard Transmittal Codes and Keyboard Layout

98-Key Keyboard Transmitted Codes					
Description	Position	Raw Code*	Hexadecimal Codes Sent		
			Unshift	Shift	Control
F1	3	20	Up to 80 programmed characters		
F2	4	26	Up to 80 programmed characters		
F3	5	05	Up to 80 programmed characters		
F4	8	3e	Up to 80 programmed characters		
F5	9	50	Up to 80 programmed characters		
F6	12	42	Up to 80 programmed characters		
F7	13	03	Up to 80 programmed characters		
F8	14	2c	Up to 80 programmed characters		
F9	15	06	ESC No	ESC NO	ESC NO
F10	16	0d	ESC Np	ESC NP	ESC NP
F11	17	67	ESC Nq	ESC NQ	ESC NQ
F12	18	7d	ESC Nr	ESC NR	ESC NR
F13	19	72	ESC Ns	ESC NS	ESC NS
ESC	25	62	1b	1b	1b
1 !	26	68	31	21	
2 @ NUL	27	6e	32	40	00
3 #	28	22	33	23	
4 \$	29	15	34	24	
5 %	30	63	35	25	
6 ^ RS	31	74	36	5e	1e
7 &	32	02	37	26	
8 *	33	08	38	2a	
9 (34	24	39	28	
0)	35	2a	30	29	
- _ US	36	30	2d	5f	1f
= +	37	2b	3d	2b	
BACK SPACE	38	64	08	08	08
DLETE	39	18	7f	7f	7f
BREAK DISCON	40	0e	break		disconnect
CLEAR RESET	41	5b	ESC [2]		ESC c
= ((pad)	42	49	3d	28	28
*) (pad)	43	11	2a	29	29
/ (pad)	44	61	2f	2f	2f

*Raw codes shown are for upstrokes of a key. Downstroke codes have the eighth bit set to 1. These codes are for use with the NOTRANSLATE mode.

Figure 11-4: 98-Key Keyboard Transmittal Codes (Sheet 1 of 3)

Keyboard Transmittal Codes and Keyboard Layout

98-Key Keyboard Transmitted Codes					
Description	Position	Raw Code*	Hexadecimal Codes Sent		
			Unshift	Shift	Control
TAB	45	4a	09	ESC [Z	ESC [Z
Q DC1	46	56	71	51	11
W ETB	47	1c	77	57	17
E ENQ	48	0f	65	45	05
R DC2	49	5d	72	52	12
T DC4	50	57	74	54	14
Y EM	51	7a	79	59	19
U NAK	52	1a	75	55	15
I HT	53	0c	69	49	09
O SI	54	1f	6f	4f	0f
P DLE	55	25	70	50	10
[{ ESC	56	10	5b	7b	1b
] } GS	57	5e	5d	7d	1d
'	59	55	60	7e	
7 (pad)	60	32	37	37	37
8 (pad)	61	12	38	38	38
9 (pad)	62	2f	39	39	39
CAPS LOCK	64	71			
CTRL	65	44			
A SOH	66	5c	61	41	01
S DC3	67	4c	73	53	13
D EOT	68	09	64	44	04
F ACK	69	69	66	46	06
G BEL	70	4b	67	47	07
H BS	71	3f	68	48	08
J LF	72	14	6a	4a	0a
K VT	73	37	6b	4b	0b
L FF	74	19	6c	4c	0c
; :	75	0a	3b	3a	
' "	76	16	27	22	
RETURN	77	6a	CR or LF or CR-LF		
FS	78	3d	5c	7c	1c

*Raw codes shown are for upstrokes of a key. Downstroke codes have the eighth bit set to 1. These codes are for use with the NOTRANSLATE mode.

Figure 11-4: 98-Key Keyboard Transmittal Codes (Sheet 2 of 3)

Keyboard Transmittal Codes and Keyboard Layout

98-Key Keyboard Transmitted Codes					
Description	Position	Raw Code*	Hexadecimal Codes Sent		
			Unshift	Shift	Control
↑	79	17	ESC [A	ESC [A	ESC [A
4 (pad)	80	1b	34	34	34
5 (pad)	81	75	35	35	35
6 (pad)	82	23	36	36	36
SCROLL LOCK	83	27			
SHIFT	84	41			
Z SUB	85	01	7a	5a	1a
X CAN	86	6f	78	58	18
C ETX	87	51	63	43	03
V SYN	88	45	76	56	16
B STX	89	33	62	42	02
N SO	90	21	6e	4e	0e
M CR	91	28	6d	4d	0d
, < NUL	92	70	2c	3c	00
. > RS	93	58	2e	3e	1e
/ ? US	94	52	2f	3f	1f
SHIFT	95	34			
ENTER	96	04	Up to 4 programmed characters		
←	97	13	ESC [D	ESC [D	ESC [D
HOME	98	31	ESC [H	ESC [H	ESC [H
→	99	4f	ESC [C	ESC [C	ESC [C
1 (pad)	100	6d	31	31	31
2 (pad)	101	6b	32	32	32
3 (pad)	102	78	33	33	33
SPACE	104	5f	20	20	20
↓	107	76	ESC [B	ESC [B	ESC [B
0 (pad)	108	53	30	30	30
. (pad)	109	2e	2e	2e	2e
F14	110	7e	ESC Nt	ESC NT	ESC NT
+ (pad)	111	7f	2b	2b	2b
- (pad)	112	73	2d	2d	2d
, (pad)	113	35	2c	2c	2c
ENTER (pad)	114	00	Up to 4 programmed characters		

*Raw codes shown are for upstrokes of a key. Downstroke codes have the eighth bit set to 1. These codes are for use with the NOTRANSLATE mode.

Figure 11-4: 98-Key Keyboard Transmittal Codes (Sheet 3 of 3)

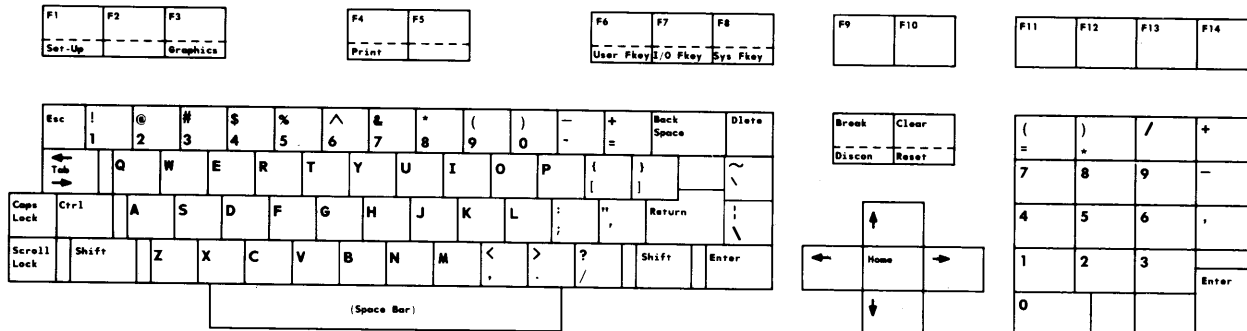


Figure 11-5: 98-Key Keyboard Layout

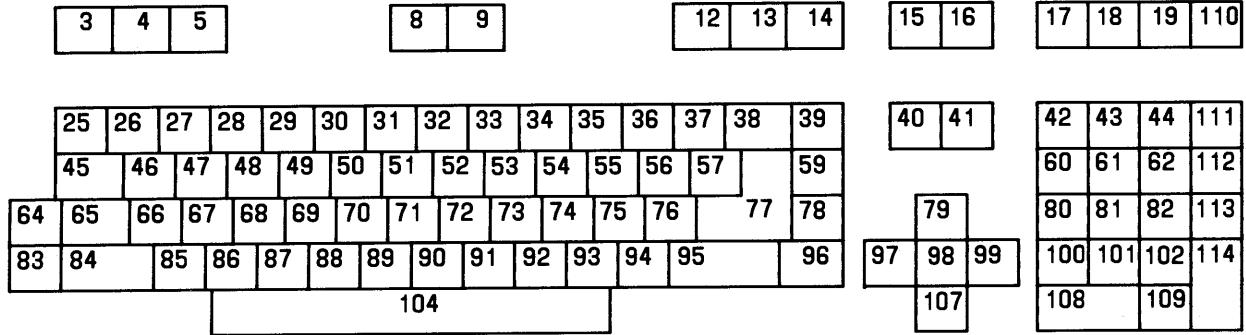


Figure 11-6: Keyboard Key Positions

Chapter 12: Dmdpi Debugger

Introduction	12-1
Dmdpi User Interface	12-2
Dmdpi Mouse Operation	12-2
Dmdpi Keyboard Input	12-4
Special Cursor Icons	12-4
Help Window	12-5
Using the Dmdpi Debugger	12-6
Dmdpi Example Demonstration	12-6
Demonstration Procedure	12-6
Other Dmdpi Features	12-25
Keyboard Expressions	12-25
Conditional Breakpoints	12-27
Spy and Journal	12-28
Assembler-Raw Memory	12-29
Dmdpi Working Directory	12-31
Changing the Dmdpi Working Directory	12-32
Changing Path to the Source Code	12-32
Debugging Crashed Processes	12-33
“clock.c” Source Code	12-35

Introduction

This chapter provides an overview of the AT&T 630 MTG Dmdpi C-Language Program Debugger. After a review of the dmdpi user interface, the user is guided through an actual debugging session.

Dmdpi ("pi" stands for process inspector) is a programmer's tool for debugging C-language programs. It allows the user to inspect and control a process (or multiple processes) in order to pin-point otherwise hard-to-find "bugs".

Dmdpi's mouse-driven, browser-like user interface allows for:

- Opening "windows" for inspecting various aspects of a process
- Controlling the execution of the process.

Dmdpi gives you power and flexibility without having to learn a complex set of commands. Through dmdpi's windows, you can get multiple "views" of a process:

- A process status/callstack window
- Several source text windows
- A global variables window
- Stack frame windows
- A breakpoint list window
- A disassembly window
- A raw memory window
- Others (user types, journal, help, etc.).

Furthermore, you can debug several processes simultaneously.

Note: The **DMDPI(1)** manual page in the "630 MTG Software Reference Manual" gives more complete coverage of the specific features of dmdpi. This chapter is not intended to be a substitute for the manual page.

Dmdpi User Interface

One of the reasons the dmdpi debugger is easy to use is its mouse-driven user interface. All operations for setting up the various process windows and selecting different functions are made with the mouse. In addition to the mouse, keyboard input is also available, where applicable.

Dmdpi Mouse Operation

Each button on the mouse performs the same basic functions regardless of which dmdpi window you are in. The following list describes these functions:

Button 1 This button "points" to make a selection in the following manner:

- Pointing at a window and clicking button 1 makes that window top and current; this is identified by a highlighted border.
- Pointing at a line of text in a current window and clicking button 1 selects that line; this is indicated by inverted video on that line.
- A scroll bar at the left of each window shows how much of the window's text is visible. Pointing into the scroll region, holding down button 1, and moving the mouse controls what text is displayed. Releasing button 1 after the desired text is located completes the operation.

Button 2 This button displays a menu of operations that apply to the current line. Positioning the cursor over the desired operation highlights that operation. Releasing button 2 selects that operation. Operations shown above the tilde line separator (~~~~~) are specific to each line. Operations below the separator are generic line operations and are listed below:

cut removes the line.
sever removes the line and all lines above it.

fold wraps a line that is past the right margin of a window onto the following line.

truncate truncates a line at the right margin.

Button 3 This button displays a menu of operations that apply to the current window. Positioning the cursor over the desired operation highlights that operation. Releasing button 3 selects that operation. Operations shown above the tilde line separator (~~~~~~) are specific to each window. Operations below the separator are generic window operations and are listed below:

reshape changes the size of the window.

move moves a window to a different place in the dmdpi window.

close deletes a window.

fold like **fold** under button 2, except it wraps every line in a window that is past the right margin onto the following line.

truncate like **truncate** under button 2, except it truncates all lines in the window at the right margin.

top gives a sub-menu which lists the currently accessible dmdpi windows. Selecting one of these windows in the sub-menu makes that window top and current. Window names appear in this sub-menu from front to back screen order; current is at the top. The **dmdpi**, **pwd/cd**, and **help** windows are always listed in this sub-menu.

Button 3 is also used to sweep out new windows, when so directed by an instruction line at the bottom of the dmdpi window. In this case, clicking button 3 gives a window of default size. A different size window can be drawn by holding down button 3 and moving the mouse.

Dmdpi Keyboard Input

The keyboard may be used in place of the mouse in many instances. Keyboard characters accumulate at the bottom of the dmdpi window. If the current line accepts input, the line flashes with each keystroke. Otherwise, if the current window accepts input, its borders flash. Characters are not interpreted until a carriage return is entered, 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

The following special cursor "icons" occasionally appear:

- arrow-dot-dot-dot** This indicates that the host is completing an operation; the terminal is ready asynchronously.
- coffee cup** This indicates the terminal is receiving input from the host; the terminal is momentarily blocked.
- exclamation mark** This indicates a dangerous menu selection. It is confirmed by pressing the menu's button again.

Help Window

A special help window, containing a reminder of the user interface mechanics, can be created from any dmdpi window by selecting **help** from the sub-menu under **top** in the window menu (button 3).

Using the Dmdpi Debugger

In the dmdpi example demonstration that follows, you will see how to:

- Open the different dmdpi windows
- Look at your program's source text and scan for specific statements and subroutines
- Set and remove program breakpoints
- Step through statements and observe the results
- Inspect local and global variables.

Dmdpi Example Demonstration

The example demonstration shows how to use dmdpi to examine a copy of the **clock** demonstration program. You will find this example most helpful if you actually follow the steps on your 630 MTG.

Remember, this example demonstration is only a training tool to familiarize you with dmdpi. For a more exhaustive description of dmdpi, consult the manual page.

While going through this demonstration, remember the usage of the three buttons on the mouse:

Button 1 makes a window or line in a window current.

Button 2 gives a line menu for the current line.

Button 3 gives the window menu and also opens new windows.

Also, the **help** window provides a review of the user interface.

Demonstration Procedure

Note: In the following demonstration, it is assumed that when instructed to input from the keyboard, you know to hit **<return>** to enter the input.

1. Execute **layers**. The 630 MTG must be operating in the **layers** environment before you can download **dmdpi**.

2. Copy the demonstration program `clock.c` into your current directory using the following command line:

```
cp $DMD/examples/clock.c exclock.c
```

While going through this demonstration, you may find it useful to refer to the printed copy of `clock.c` at the end of the chapter.

3. Compile `exclock.c` using the following command line:

```
dmdcc -g -o exclock exclock.c
```

The executable object code will be called `exclock`. The `-g` option generates the symbol table and debugging information that is needed by `dmdpi`.

4. In a new window, download the compiled `exclock` file using:

```
dmdl -z exclock "`date`"
```

The `-z` option prevents the program from executing after being downloaded. This will allow you to take control of the `exclock` process with `dmdpi` before it starts running.

In real debugging situations, the `-z` option is often useful for debugging programs that crash when the program first boots after download. When the program is downloaded with `-z`, you can use `dmdpi` to take control of the process in order to set breakpoints, look at variables, etc., before the crash occurs.

The `"`date`"` evaluates the UNIX System V `date` function and sends this to the program as an argument. `exclock` will use this to set the current time.

Note: The `date` command should be enclosed by grave accent marks rather than single quotes.

5. In a new window, download `dmdpi` by typing the following command. (Make sure that you are in the same directory as `exclock`.)

```
dmdpi
```

Note: For the purpose of this demonstration, make the size of this window about one-third of the screen. (Once you become familiar with `dmdpi`, you can decide what size window works best for you.)

Using the Dmdpi Debugger

The directory where **dmdpi** is executed becomes dmdpi's initial working directory. By default, dmdpi searches its working directory for the object code and source code files of the process being debugged. In real debugging situations, however, these files may not be in dmdpi's initial working directory. The "Dmdpi Working Directory" section (in "Other Dmdpi Features") explains what to do in these situations.

Changing directories will not be a problem in the following dmdpi demonstration, since the **exclock** process and **dmdpi** have been downloaded from the same directory.

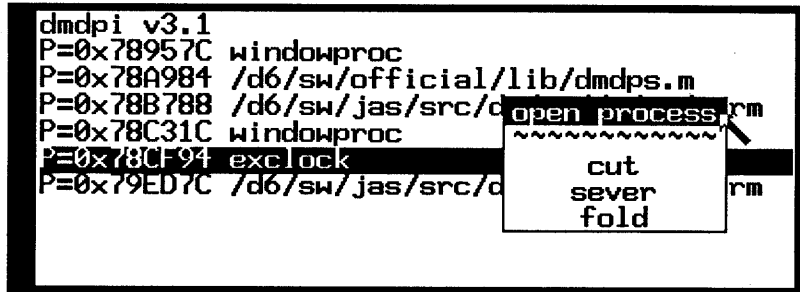
Dmdpi Window

6. Once **dmdpi** has finished downloading, you will be instructed to sweep a dmdpi window (button 3). If **dmdpi** is not running in the current window, a small dmdpi window will be created automatically.

```
dmdpi v3.1
P=0x78957C windowproc
P=0x78A984 /d6/sw/official/lib/dmdps.m
P=0x78B788 /d6/sw/jas/src/dmdpi/dmdpi/term
P=0x78C31C windowproc
P=0x78CF94 exclock
P=0x79ED7C /d6/sw/jas/src/dmdpi/dmdpi/term
```

The dmdpi window gives the listing of all processes currently running in the terminal. Each line gives the hexadecimal key and pathname to the executable file on the host that is associated with that process. Dmdpi uses the pathnames in this window to locate host resident symbol tables for processes being debugged.

- Point to the **exclock** process line and make it current (button 1).
Select **open process** from the line menu (button 2).



You will be prompted to open a process control window.

Process Control Window

The process control window (sometimes called process window) controls exactly one process. It allows you to start and stop the execution of a process. The process control window displays the state of the process, and allows you to create process inspection windows to see more detailed views of the process.

The process control window will also display a callstack trace when a process is stopped. The callstack trace gives a list of functions that were called in order to reach the location where the process stopped. This is the dynamic chain of activation records.

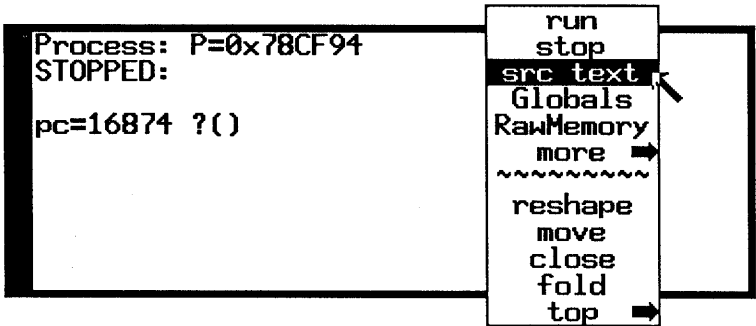
- As instructed, open a process control window (button 3).

```
Process: P=0x78CF94
STOPPED:

pc=16874 ?()
```

Notice that the state of the process is STOPPED.

- 9. In the process control window, select **src text** under the window menu (button 3).

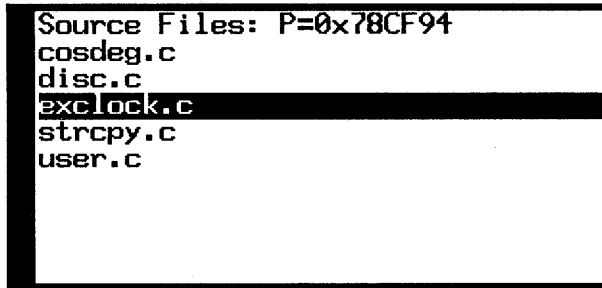


You will be prompted to open a source files window.

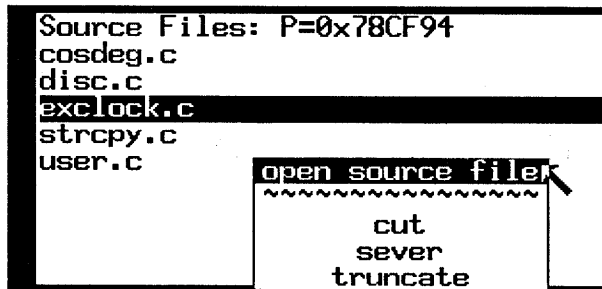
Source Window(s)

- 10. Open a source files window (button 3). This window lists all the source files associated with the demonstration program. Make the **exclock.c** line in that window current (button 1).

Note: If there has been only one source file associated with **exclock**, you would have been prompted to open a source text window (step 12).



11. Select **open source file** from the line menu (button 2).



You will be prompted to open a new window.

12. Open a source text window (button 3). This window displays the source code of the demonstration program.

Note: Since this window displays the actual lines of the program source code, you may want to make this window larger than the default window size.

```
Source Text: exclock.c
/* Copyright (c) 1987 AT&T */
/* All Rights Reserved */

/* THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF
/* The copyright notice above does not evidence a
/* actual or intended publication of such source

#include <scsid.h>
VERSION(@(#)clock.c 1.1.1.1);

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

You probably noticed that when the source text window was first opened, only the line numbers appeared, then the source was displayed. This is because the source text window works on a "per request basis." Opening the window initiated a request to the host to download only the lines of source on the screen which, in this case, are the first lines of source. As you request more lines of source (by scrolling or searching through the text window), additional lines will be downloaded and displayed. Once the lines of source have downloaded, they remain in local memory as long as this text window remains open.

13. Point to the scroll bar (left margin of the source text window). With button 1 depressed, move the scroll bar down. As you move the mouse, you should be scrolling through lines of the source file (lines may only be numbers initially).

From the keyboard, you can initiate a search for a specific data pattern in the file (pattern should be preceded by a slash "/").

You can also go to a specific line number. In the source window with no line current (scroll the pointer off the window), type in the desired line number.

To go to a subroutine in the program, just select the subroutine from

the list in the window menu (button 3).

Select the subroutine **main** from window menu (button 3).

```

Source Text: exclock.c
/* Copyright (c) 1987 AT&T
/* All Rights Reserved

/* THIS IS UNPUBLISHED PROPRIETARY
/* The copyright notice above
/* actual or intended publication rights are hereby
   reserved.

#include <ccsid.h>
VERSION(("@(#)clock.c 1.1.1.1)

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

```

step 2 stmts
step 3 stmts
step 4 stmts
step >4 stmts → OF
step into fcn → a
interface()...128 ce
main().....31 ←
ray().....154
~~~~~
reshape
move
close
fold
    
```

Breakpoints

Dmdpi allows you to place breakpoints in the source text of a program. When the program reaches a breakpoint during execution, it stops, and control is given to the debugger. At this point, dmdpi can be used to inspect and/or modify program variables or to resume execution.

In dmdpi, when a breakpoint is set on a certain statement, this statement does not get executed. In other words, execution "breaks" after the last statement before the breakpoint.

14. From the previous step, the process should be at the subroutine **main**. Make the line containing the **rectf** statement below **main** current (button 1). In the line menu (button 2), select **set bpt**. This sets a breakpoint at the **rectf** statement.

Notice that >>> appears at the beginning of the source line to indicate a breakpoint has been set.

```

register long ds;
register olds;

/* do initialization */
oldtime=realtime();
request(KBD);
rectf(&display, Direct, F_XOR);
if(argc!=2){
    jmoveto(Pt(0, 0));
    jstring("Usage: dmdld clock
           sleep(200);
           exit();
        }
    }

```

```

set bpt
trace on
cond bpt
assembler
open frame
~~~~~
cut
sever
fold
    
```

15. From the window menu again (button 3), go to the `initface` subroutine and set another breakpoint (button 2) at the `disc` statement below `initface`.

```

    rad = Direct.corner.y - Direct.origin.y;
    rad = rad/2 - 2;
    rh = 6 * rad / 10;
    rm = 9 * rad / 10;
    rs = rad - 1;
    rspread = rad / 10;
    >>> disc(&display, ctr, rad, F_STORE);
    first = 1;
}

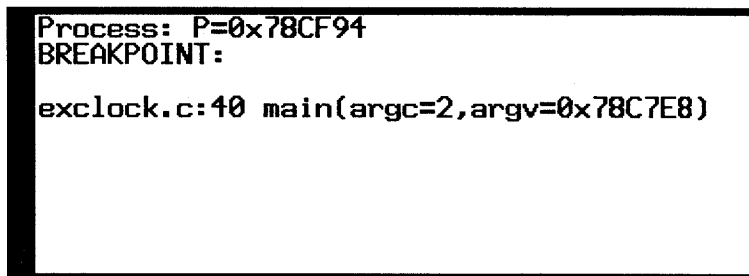
/* draw ray r at angle ang.
/* actually draws a clock hand at angle ang, with

```


Call Stacks and Stack Frames

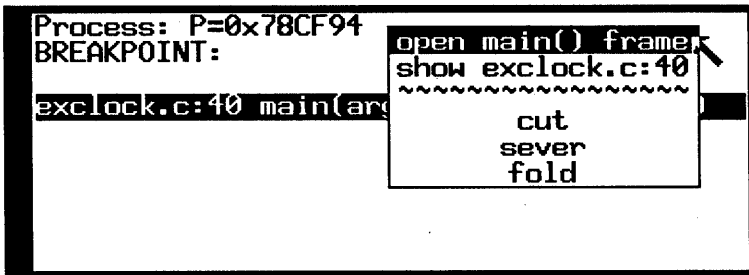
16. From the window menu (button 3) in the source text window, select the **run** item. The program will run until it hits the first breakpoint at the **rectf** statement in the **main** subroutine. The program stops at the breakpoint but does not execute the **rectf** statement.

Notice that when the program stops at the breakpoint, the contents of the process control window changes. The state now reads **BREAKPOINT**, the callstack shows that the program is in **main**, and there is one item on the callstack.



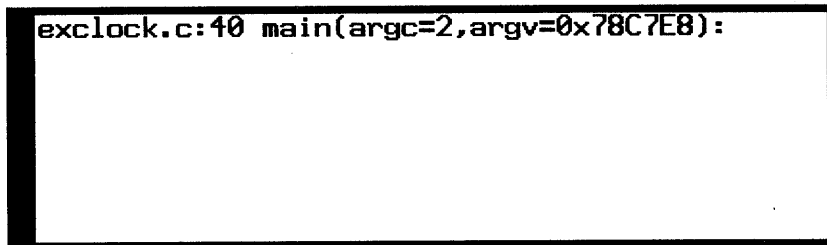
```
Process: P=0x78CF94  
BREAKPOINT:  
exclock.c:40 main(argc=2,argv=0x78C7E8)
```

17. Make the process control window current and highlight the line for **main** in the callstack. From the line menu (button 2), select **open main() frame**.



You will be prompted to open a new window. The new window is called a stack frame window and is illustrated below.

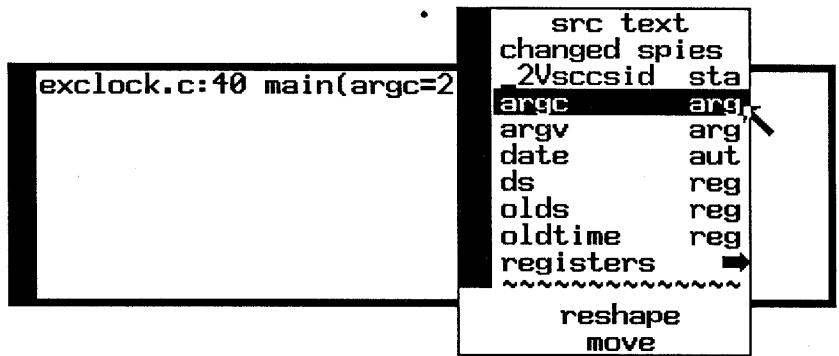
Note: You could also have opened the stack frame window from the source text window by selecting **open frame** from the line menu (button 2).



A stack frame window allows inspection of a subroutine's activation record which contains information such as passed parameters, local variables and register contents. Each subroutine on the callstack has a corresponding activation record and, therefore, each can have a corresponding stack frame window.

Local Variables

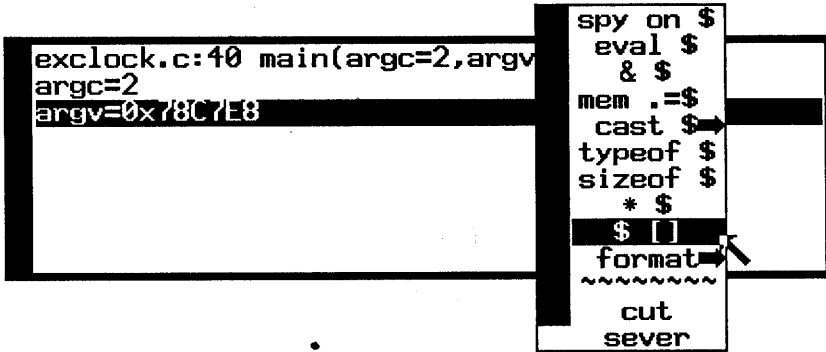
- From the stack frame window menu (button 3), select and look at the current values of local variables `argc` and `argv`.



- Now the contents of the variables `argv[0]` and `argv[1]` can be displayed. From the line menu (button 2) on `argv` (button 2), select `$[]`.

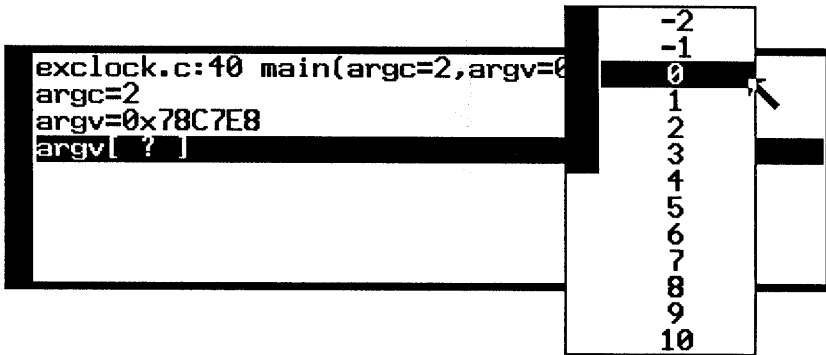
The operator `$` always refers to the current (highlighted) expression. Therefore, the `$[]` menu item says that you want to look at the array contents of the current expression (`argv`).

Using the Dmdpi Debugger



This menu item generates a new line, `argv [?]`, in the stack frame window.

20. The line menu (button 2) for `argv [?]` displays a list of values that you can set `argv` equal to. Select a value of 0.



Now select a value of 1. Notice the results in the stack frame window.

```

exclock.c:40 main(argc=2,argv=0x78C7E8):
argc=2
argv=0x78C7E8
argv[ ? ]
argv[1]=0x78C7FC="Wed Mar 25 12:10:44 ..."
argv[0]=0x78C7F4="exclock"

```

21. Highlight the `argv[1]` line and select "wider" on under format of the line menu (button 2). This will give you the full string of `argv[1]`.

The screenshot shows the debugger's stack frame window with the following content:

```

exclock.c:40 ma
argc=2
argv=0x78C7E8
argv[ ? ]
argv[1]=0x78C7F
argv[0]=0x78C7F

```

The line `argv[1]=0x78C7F` is highlighted. A context menu is open over it, showing the following options:

```

spy on $
eval $
& $
mem .=$
cast $
typeof $
sizeof $
* $
$ []
hex off
unsd_dec on
sign_dec on
octal on
ascii on
float on
"ascii" off
"wider" on
symbolic on

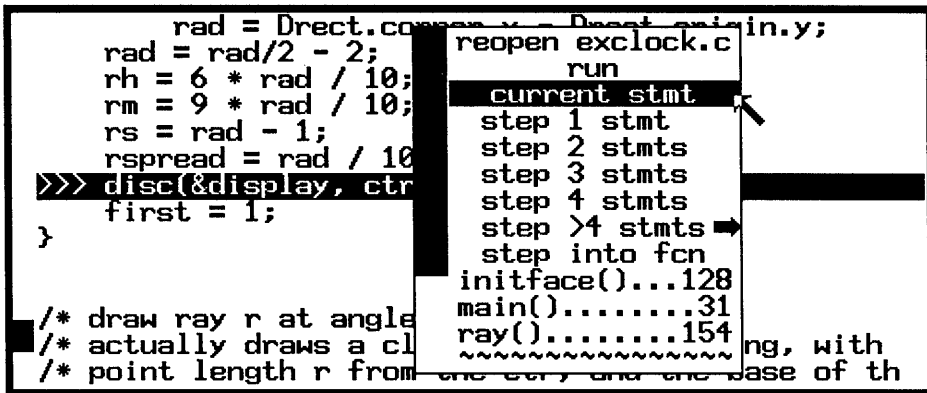
```

The "wider" option is selected, and a mouse cursor is pointing at it.

You may want to take time at this point to look at some of the other items associated with the stack frame window.

Step Statements

22. Make the source text window current. With the **current stmt** item under the window menu (button 3), verify that the program is still at the first breakpoint (**rectf**). (The **current stmt** menu item highlights the current statement in the source text window.)



Under the same menu (button 3), select **step 2 stmts**. The program will stop at **local()**. Notice that the **rectf()** function executed and the **exclock** window was filled with reverse video.

Step 1 statement: notice the **exclock** window becomes a local window (checkered border).

23. The program should now be at line **initface**. This line initiates the call to **initface**. You can enter this function by the **step into fcn** window menu item. However, since there is already a breakpoint at **disc** in this function, just select **run**. The program will stop at the breakpoint. Notice that a new item is displayed at the top of the call stack in the process control window.

```
Process: P=0x78CF94  
BREAKPOINT:  
  
exclock.c:142 initface()  
exclock.c:49 main(argc=2,argv=0x78C7E8)
```

24. Step 3 statements: you should get the clock face on the clock window which was drawn by the call to **disc**. Notice that the **initface** subroutine has returned; therefore, **initface** is no longer in the call stack. The program should stop at **strcpy**.
25. From the line menu (button 2), select **open frame**. This will make the stack frame **main** current. (If you removed the stack frame window, you will be prompted to open a new one.) Look at the value of **date** from the window menu (button 3). Step the program one statement (executing **strcpy**) and then look at the values of **date** again.

```
exclock.c:50 main(argc=2,argv=0x78C7E8):  
date=0x78DA74=""  
date=0x78DA74="Wed Mar 25 12:10:44 ..."
```

Global Variables

- 26. To display global variables, select the **Globals** item from the window menu (button 3) of the process control window. You will be prompted to open a Globals window.

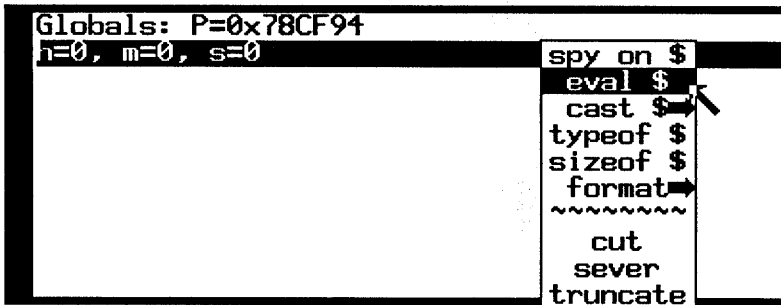
The values of global variables can be displayed by either choosing the variable from the Globals window menu (button 3) or by typing an expression from the keyboard.

Remember that each line or window that accepts keyboard input produces some help in response to typing ?. Type ? to see the help message for the Globals window. The help message says that an expression can be entered.

Type in the following expression for variables **h**, **m**, and **s**.

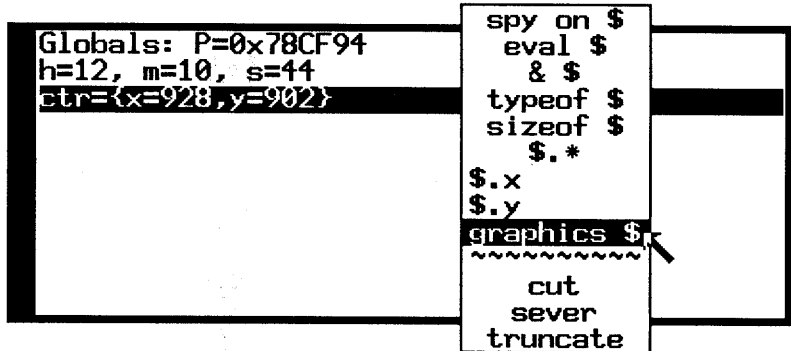
h, m, s

Now step the program 3 statements and look at these variables again. This time use the **eval \$** menu item (button 2) in the Globals window (the variables line has to be highlighted). The **eval \$** menu item re-evaluates the current expression.



As you might have guessed, these variables are the hours, minutes, and seconds for the clock program.

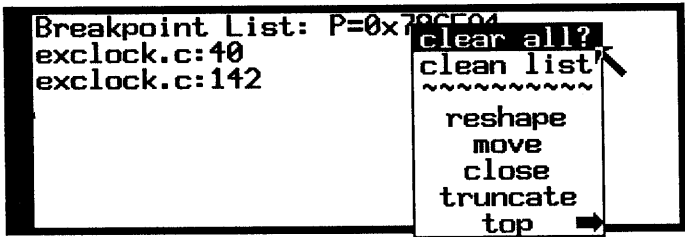
27. Look at the value of `ctr` in the globals window. This is the center point on the clock window. Select **graphics \$** from the line menu (button 2). This shows where the center point is in the clock window. To exit this mode, just click any button on the mouse.



Clearing Breakpoints

Before you begin running the program, you need to go back and clear the breakpoints.

28. The easiest way to remove all breakpoints is from the process control window. Select **Bpt List** from the sub-menu under **more** of the window menu (button 3). You will be prompted to open a Breakpoint List window. Select the **clear all?** item from the window menu (button 3). An exclamation mark inside a diamond (which indicates a dangerous menu selection) will be displayed. This is considered dangerous only because **all** breakpoints will be removed, which is something you may not want to do in all situations. Clearing all breakpoints is confirmed by pressing the window menu's button (3) again.



29. Breakpoints can be individually removed by using the **clear bpt** in the line menu (button 2) with the desired breakpoint line highlighted in the Breakpoint list window. A breakpoint can also be removed from the source text window with the **clear bpt** item (button 2) and the appropriate source line highlighted.
30. You can now start the program executing. From either the process control or source text window, select **run** from the window menu (button 3). The clock window will graphically display a clock.

Note: The time is not correct because some time has elapsed since **date** was executed on the host (when **exclock** was downloaded).

In the following section, "Other Dmdpi Features," you will see how to reset the clock to the correct time plus other features of dmdpi that were not covered in this initial section of the demonstration.

Other Dmdpi Features

This section gives examples of using other dmdpi features that were not included in the previous section for the dmdpi demonstration. These features are:

- Keyboard Expressions
- Conditional Breakpoints
- Spy and Journal
- Assembler-raw memory
- Pathnames/dmdpi working directory
- Crashed processes.

This section again uses the **exclock.c** demonstration for its examples. If you need help in downloading and setting up the dmdpi windows for **exclock.c**, refer to the previous section, "Using the Dmdpi Debugger-Example Demonstration."

Keyboard Expressions

From the previous section, you saw how to use the stack frame and Globals windows to display the values of local and global variables. These windows can also be used to evaluate valid C-language expressions that are entered from the keyboard.

1. Open a Globals window from the process control window.

You may use this window to perform calculations. Enter each of the following expressions from the keyboard:

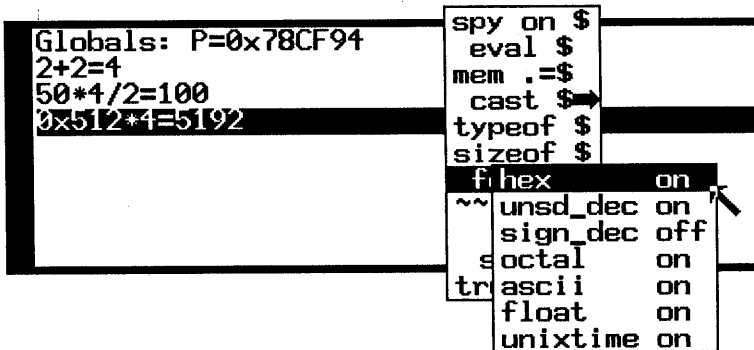
2+2

50*4/2

0x512*4

Other Dmdpi Features

Look at the hexadecimal value of $0x512*4$ by selecting the **hex on** item from the sub-menu under **format** in the line menu (button 2).



2. Use the window to evaluate other expressions. Enter each of the following ascii expressions:

`'a'`

`'9'-'0'`

3. Look at the values of variables **h**, **m**, and **s**. Enter the following expressions to change the contents of **h**, **m**, and **s**. (Notice the results in the clock window.)

`h=6`

`m=30`

`s=0`

The following expression is also valid:

`m=h*2`

4. Now set the correct time in the clock window.

Conditional Breakpoints

A conditional breakpoint can be inserted into the source code to test for a certain condition. When the condition is true, the program's execution is stopped at the breakpoint.

1. To demonstrate this feature in the **exclock** example, search for the following line in the source text window:

```
s += ds / 60; /* calculate seconds */
```

This line calculates the seconds.

2. Select **cond bpt** from the line menu (button 2). This places an **if(?) >>>** expression in front of the line. The (?) is where the condition is to be defined.

For the condition, we want to initiate a breakpoint when the second hand in the clock window is between the 55- and 5-second points on the clock face. In order to see all that happens, type in the following expression and wait until the second hand has passed the 5-second point before hitting <return>. This allows you to see the condition being installed, the condition becoming true, and the results of the breakpoint.

```
s, s>55!|s<5
(passed 5-second point?)
hit <return>
```

A series of expressions may be entered, separated by the comma-operator. The value of each sub-expression will be printed, but it is the value of the last expression that sets the condition for the breakpoint. In the example, the sub-expression **s** will be evaluated and printed each time the conditional breakpoint is evaluated. However, execution of the clock will stop only when the value of the last expression (**s>55!|s<5**) becomes true (not equal to 0).

```
for ( ds = 0;; ) {
    /* process times */
    while (realtime() <= oldtime)
        sleep(20);
    ds += realtime()-oldtime; /* elapsed tick
    oldtime=realtime();
    if( (14) s=50, s>55; s<5=1 )>>> s += ds / 60;
    ds %= 60; /* left over ticks */
    if (olds == s)
        continue;
    /* calculate seconds, minutes and hours */
    while (s >= 60) {
        s -= 60;
        m++;
    }
}
```

3. Now clear the breakpoint (**clear bpt** from line menu - button 2) and select **run** from the window menu (button 3).

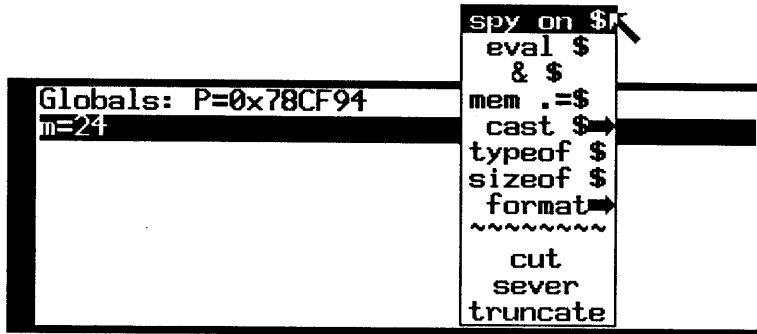
The **trace on** item under the source text line menu (button 2) sets a conditional breakpoint of 0 that will never be true. This serves the purpose of tracing the statement without stopping execution on the breakpoint.

Spy and Journal

Spy allows you to observe changes in the value of an expression. The value of the expression is evaluated each time dmdpi looks at the expression. If the value of the expression has changed, then dmdpi will notify you of that fact.

Note: The value of the expression may change several times before dmdpi actually gets a chance to look at the process. More specifically, dmdpi will only see the expression change when the process being debugged calls `wait()` or `sleep()`. Therefore, you may not see every change in the value of an expression.

1. In the Globals window, look at the current value of **m**. From the line menu (button 2), select **spy on \$**.



When the second hand on the clock passes 60, the value of **m** changes and dmdpi flashes a message in the process window and updates the value of **m** in the Globals window.

2. The **Journal** window (from the process control window under the **more** sub-menu) keeps a record of major debugger activities that are taking place in the process. For example, while "spying" on the variable **m**, the **Journal** window keeps record of the state of the process and each **spy** update.
3. Remove the **spy on \$** the same way it was initiated (from the Globals window's line menu (button 2) with **m** line current), except this time the line menu item says **unspy \$**.

Assembler-Raw Memory

The Assembler window displays the assembler code for statements that you select.

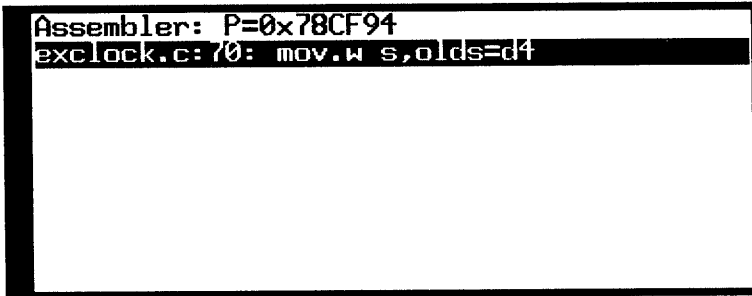
1. In the source text window for **exclock**, highlight the following line of code.


```
olds = s;
```
2. Select the **assembler** item from the line menu (button 2).

Open the assembler window.

Other Dmdpi Features

3. In the Assembler window, observe the assembler code for this statement.

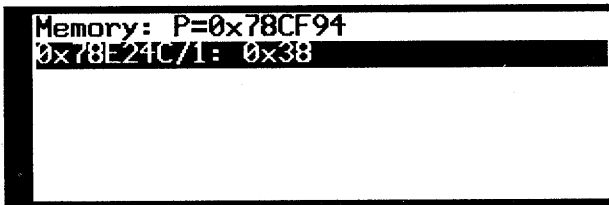


```
Assembler: P=0x78CF94
exclock.c:70: mov.w s,olds=d4
```

From the line menu (button 2), you can also look at the assembler code for the next instructions using the **next** menu items (**next 1**, **next 5**, etc.).

4. With the **mov.w** line highlighted, select **raw mem** from the line menu (button 2).

Open the Memory window. This window displays the actual contents of the memory locations.



```
Memory: P=0x78CF94
0x78E24C/1: 0x38
```

This is the first byte of the **mov.w** opcode.

5. Selecting **+.1** from the line menu (button 2) gives the next memory location. (0x3839 is the full opcode for the **mov.w** instruction.)

Dmdpi Working Directory

As discussed earlier, the directory where dmdpi is executed becomes dmdpi's initial working directory. Dmdpi's working directory is important when opening process control and source windows.

When opening a process control window, if a process is downloaded without a full pathname, dmdpi will look for the object module in its working directory. For example, if a process is downloaded with the following command line:

```
dmdld -z demo.m
```

the process is listed in the dmdpi window as **demo.m**. When you open a process control window for that process, dmdpi (by default) looks for the **demo.m** object file in the dmdpi working directory. If dmdpi cannot find the object file, which contains the symbol table and debugger information, the following message is displayed in the process control window:

```
Process: P=0x78F064  
STOPPED:  
symbol table header: Cannot find symbol  
table (try pwd/cd); go on  
pc=16874 ?()
```

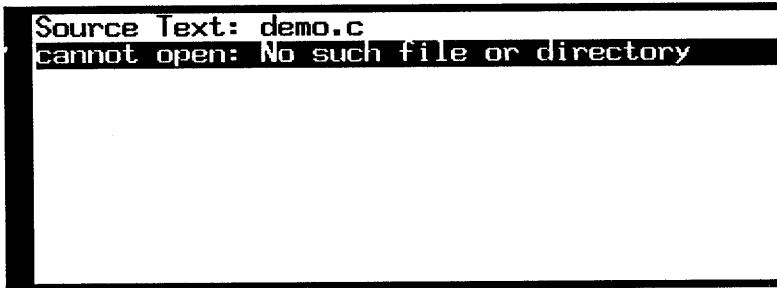
The process, however, may be downloaded with a full pathname such as in the following command line.

```
dmdld -z /usr/tac/demo/demo.m
```

In this case, the process will be listed in the dmdpi window as **/usr/tac/demo/demo.m**. Since dmdpi now knows the full pathname to the object module, the object module can be located when a process control window is opened for that process regardless of dmdpi's working directory.

Other Dmdpi Features

When opening a source file window, dmdpi will look for the source file in its working directory. If dmdpi cannot find the source file, the following message will be displayed in the source text window:

A screenshot of a source text window. The window has a black border and a white background. At the top, the text "Source Text: demo.c" is displayed. Below it, a black horizontal bar contains the text "cannot open: No such file or directory" in white. The rest of the window is empty.

```
Source Text: demo.c
cannot open: No such file or directory
```

Changing the Dmdpi Working Directory

To change the dmdpi's working directory, do the following:

1. In any window, you may select **pwd/cd** sub-menu item of **top** in the window menu (button 3).
2. Sweep a working directory window (button 3).
3. From the keyboard, type in the name of a directory to change dmdpi's working directory. This enables dmdpi to locate the object and source code. Selection of a new directory may also be made with the mouse.

Changing Path to the Source Code

In order for dmdpi to find the source code of a process without changing its working directory, you may enter a textual prefix into the source files window. (This can only be done if there is more than one source file.)

1. From the process control window, select **src text** from the window menu (button 3). Sweep the source text window. If there is more than one source file, a window listing the file names will appear.
2. From the keyboard, type in a directory name where dmdpi can find these source files.
3. A line at the top of the window will appear containing this prefix. Dmdpi will look for these files in this directory rather than looking for them in its working directory.

Debugging Crashed Processes

If a process crashes, one of the major steps in determining the problem is locating where it crashed. Dmdpi can help do this.

A program crashing commonly causes a process exception. When this happens, a message will be displayed at the bottom of the application's window. At this point, the crashed process can be opened for debugging by dmdpi, and a callstack traceback can be generated. The callstack traceback can be used to find the line of source text where the exception occurred.

The following program will crash when it is downloaded.

```
main()
{
    int *loc = 0;

    *loc = 0;
    sleep(2);
    .
    .
    .
}
```

This code attempts to write the value 0 to memory address 0. Since the 630 MTG has ROM memory at address 0, this code will cause a bus error process exception.

Other Dmdpi Features

To observe how dmdpi can be used to determine where a program crashed, do the following:

1. Type the previous code into a file, compile it using **dmdcc** with the **-g** option, and download it with **dmdld**. After it has finished downloading, notice the PROCESS EXCEPTION message in the application's window.
2. Choose the crashed process for debugging and open a process control window. The process control window will indicate that a process exception has occurred, and a callstack traceback will be generated showing where the process was executing when the exception occurred.
3. Now you can determine exactly which line of source code caused the process to crash. In the process control window, make the callstack traceback line for function **main()** current.

Note: In the example, there is only one line in the callstack; however, in real debugging situations, there may be several lines in the callstack. In these situations, the current function will be the top line on the callstack.

Select **show filename** from the line menu (button 2). You will be prompted to sweep a source text window. In the source text window, the line containing **sleep(2)** will be highlighted. Since **sleep(2)** is the next line to execute, this implicates the previous line, ***loc=0**, as the problem.

“clock.c” Source Code

The following is a printout of the source code for the `clock.c` program that was used in the example demonstration of `dmdpi`.

```
/*      Copyright (c) 1987 AT&T      */
/*      All Rights Reserved      */

/*      THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF AT&T      */
/*      The copyright notice above does not evidence any      */
/*      actual or intended publication of such source code.      */

#include <sccsid.h>
VERSION(@(#)clock.c 1.1.1.1);

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

#define atoi2(p) ((*(p)-'0')*10 + *((p)+1)-'0')
#define itoa2(n, s) { (*(s) = (n)/10 + '0'); *((s)+1) = n % 10 + '0'; }

extern Point string();
extern Point add();
extern long realtime();

Point ctr; /* center point of window */
int h, m, s; /* hour, min, sec */
int rh, rm, rs; /* radius of h m and s */
int rspread; /* spread of base of hands at ctr */
int ah = 0, am = 0, as = 0; /* angles based on 360 degrees of circle */
int lah, lam, las; /* last angles */
int first ;

main(argc, argv)
char *argv[];
{
    char date[40];
    register long oldtime;
    register long ds;
    register olds;

    /* do initialization */
    oldtime=realtime();
    request(KBD);
    rectf(&display, Drect, F_XOR);
}
```

"clock.c" Source Code

```
if(argc!=2){
    jmoveto(Pt(0, 0));
    jstring("Usage: dmdld clock
sleep(200);
exit());
}
local();
initface();
strcpy(date, argv[1]);
h = atoi2(date+11);
m = atoi2(date+14);
s = atoi2(date+17);

/* main loop for clock */
for ( ds = 01; ) {
    /* process times */
    while (realtime() <= oldtime)
        sleep(20);
    ds += realtime()-oldtime; /* elapsed ticks (1/60 sec) */
    oldtime=realtime();
    s += ds / 60; /* calculate seconds */
    ds %= 60; /* left over ticks */
    if (olds == s)
        continue;
    /* calculate seconds, minutes and hours */
    while (s >= 60) {
        s -= 60;
        m++;
    }
    olds = s;
    while (m >= 60) {
        m -= 60;
        h++;
        if (h >= 24)
            h = 0;
    }

    /* save old angle */
    lah = ah;
    lam = am;
    las = as;
    /* calculate new angles */
    ah = (30 * (h%12) + 30 * m / 60);
    am = 6 * m;
    as = 6 * s;
```

```
if (P->state & RESHAPED) {
    initface();
    P->state &= ~RESHAPED;
}
if ( kbdchar() == 'q' )
    exit();

/* write digital time to display */
strcpy(date, "00:00:00");
itoa2(h, date);
itoa2(m, date+3);
itoa2(s, date+6);
string(&mediumfont, date, &display, add(Direct.origin, Pt(1,1)),
    F_STORE);

/* draw the hands of the clock */
if(first) {
    ray(rs, as, rspread/2);          /* longest */
    ray(rm, am, rspread);
    ray(rh, ah, rspread);
    first = 0;
}
else {
    if(lah != ah) {
        ray(rh, lah, rspread);
        ray(rh, ah, rspread);
    }
    if(lam != am) {
        ray(rm, lam, rspread);
        ray(rm, am, rspread);
    }
    ray(rs, las, rspread/2);
    ray(rs, as, rspread/2);
}
}
```

“clock.c” Source Code

```
/* set up clock circle in window */
initface()
{
    int    rad;

    rectf(&display, Drect, F_CLR);
    ctr.x = (Drect.corner.x + Drect.origin.x) / 2;
    ctr.y = (Drect.corner.y + Drect.origin.y) / 2;
    rad = Drect.corner.x - Drect.origin.x;
    if (rad > Drect.corner.y - Drect.origin.y)
        rad = Drect.corner.y - Drect.origin.y;
    rad = rad/2 - 2;
    rh = 6 * rad / 10;
    rm = 9 * rad / 10;
    rs = rad - 1;
    rsprad = rad / 10;
    disc(&display, ctr, rad, F_STORE);
    first = 1;
}

/* draw ray r at angle ang. */
/* actually draws a clock hand at angle ang, with the hand at a */
/* point length r from the ctr, and the base of the hand spread */
/* by rspr. */

ray(r,ang,rspr)
register int r, ang, rspr;
{
    register int dx, dy;
    register int ddx1, ddx2, ddy1, ddy2;

    dx = muldiv(r, Isin(ang), 1024);
    ddx1 = muldiv(rspr, Isin(ang-90), 1024);
    ddx2 = muldiv(rspr, Isin(ang+90), 1024);
    dy = muldiv(-r, Icos(ang), 1024);
    ddy1 = muldiv(-rspr, Icos(ang-90), 1024);
    ddy2 = muldiv(-rspr, Icos(ang+90), 1024);
    segment(&display, add(ctr, Pt(ddx1,ddy1)), add(ctr, Pt(dx,dy)), F_XOR);
    segment(&display, add(ctr, Pt(ddx2,ddy2)), add(ctr, Pt(dx,dy)), F_XOR);
}
}
```

Chapter 13: Jim Text Editor

Introduction	13-1
The Jim Window	13-2
Jim Operations	13-4
Using the Mouse in Jim	13-4
Getting Started - Creating a Text Frame	13-4
Loading Files into "jim"	13-5
Selecting Text	13-6
Positioning the "text cursor"	13-6
Selecting Words	13-6
Selecting a Line	13-7
Selecting an Arbitrary Continuous Text String	13-7
Deleting Text	13-7
Deleting Text with the Mouse	13-7
Deleting Text Using the Keyboard	13-8
Adding New Text	13-8
Moving and Copying Text	13-8
Moving or Copying Text within a Frame	13-8
Moving or Copying Text to Another Frame	13-9
Changing Text	13-9
Save Buffer Considerations	13-10
Write Commands	13-10
Writing a File Using Button 3	13-10
Writing Files with the "w" Command	13-11
Naming Files	13-11
Naming or Renaming a File	13-11
Determining the Filename of a Current Frame	13-12
String Searches	13-12
Other jim Commands	13-13
Editor Commands	13-13
Miscellaneous Commands	13-13
I/O Redirection Commands	13-13
Frame Positioning and Scrolling	13-14

Chapter 13: Jim Text Editor

Positioning Cursor from the Active Command Line 13-14
Using the Mouse to Position the Cursor 13-14
Scrolling within a Frame 13-14
Special Characters 13-15
Button 3 Menu Features 13-15
Quitting Jim 13-17
Command Line Diagnostic Messages 13-17
Interactively Recovering Lost Files 13-22

Keyboard Command Summary 13-24

Introduction

The **jim** visual text editor allows you to use a mouse to manipulate text in a file. The mouse can be used to copy, move, or delete text. Several files can be edited at the same time and data can be transferred easily between the files being edited. The **layers** environment must be loaded before you can use **jim**.

You will find **jim** easy, fast, and fun to use. It may, however, take some practice to become acclimated to using the mouse as an editing device.

The instructions in this chapter for using **jim** are presented in a tutorial sequence. First-time users should try each step while they are reading the tutorial. After gaining familiarity with **jim** operations, use the **JIM(1)** manual page in the *630 MTG Software Reference Manual* for quick reference.

The Jim Window

The **jim** window is the **jim** application program downloaded into the **layers** environment. After the program is loaded, a one-line command and diagnostic frame will appear at the bottom of the **jim** window. This frame can be used to enter **jim** commands. It will also display diagnostic messages (see "Command Line Diagnostic Messages" at the end of this chapter).

You may open one or more multiline text frames inside the **jim** window. These frames are used for viewing and editing files on the host computer. The size of each frame is determined by the user. Each frame has a scroll bar, a positional "tick," and a movable "text cursor."

Note: The maximum number of frames that can be made is dependent on file sizes and the amount of 630 MTG memory that is available.

The **jim** editor can be downloaded into more than one window. However, first-time users should download **jim** into only one window to avoid confusion. The frames in **jim** are made and reshaped similarly to windows.

Figure 13-1 shows a **jim** window with three frames. Two other windows are shown for demonstration purposes.

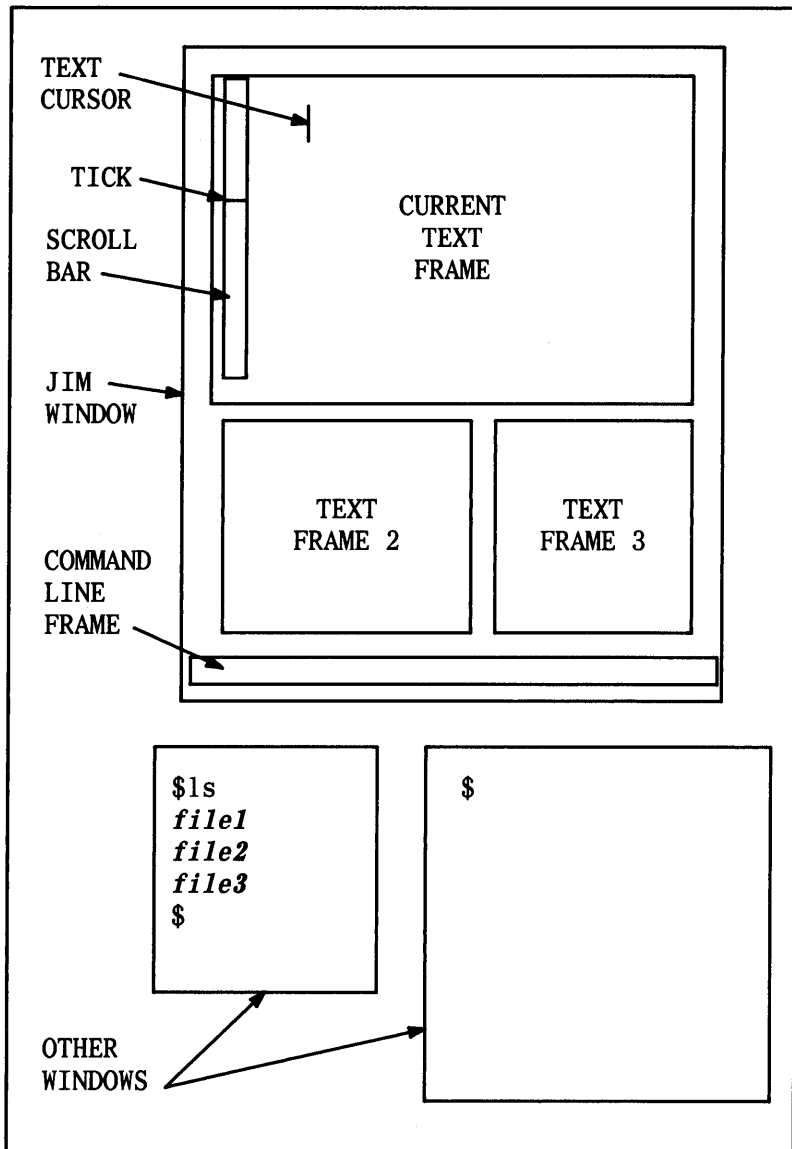


Figure 13-1: Sample 630 MTG Screen with Three Jim Frames Open

Jim Operations

When using **jim** for the first time, it is suggested that you use “junk” files for practice.

Using the Mouse in Jim

In general, button 1 is used for text selection, button 2 provides a menu of text manipulation functions, and button 3 provides control for frame operation.

Avoid moving the mouse and clicking button 2 or button 3 at the same time. You may accidentally select a menu item that you did not mean to select. If button 2 or button 3 is depressed accidentally, move the cursor outside the menu and then release the button to avoid selecting a menu item.

Getting Started - Creating a Text Frame

1. In the **layers** environments, create a window about two-thirds the height of the screen.
2. Type **jim** and press **Return**. The **jim** editor program will begin to download into the window. When the download is completed, the command line frame will appear at the bottom on the window.
3. Move the arrow cursor inside the **jim** window.
4. Press (and hold) button 3. The **jim** command menu will appear.
5. Point the cursor at **new** in the menu (inverse video) and then release button 3. This makes the menu selection to create a new frame. The arrow will change to a sweep cursor (small square with an arrow). (Clicking button 1 or button 2 cancels this request for a new frame.)
6. Position the sweep cursor in the area of the **jim** window where one corner of the new frame will be. Press (and hold) button 3 and move the mouse, forming a rectangular frame the size that you desire. For demonstration purposes, make a text frame that covers the lower one-half of the **jim** window. This frame will have an inverse video scroll bar to indicate that it is the current text frame.

Note: Before any additional frames can be opened, the frame created last must receive some type of input; such as a character typed in it, a file loaded, etc.

7. You can now start typing text into this new frame. Try it; type in your favorite quote. Later in this chapter, you will see how to change, delete, and add text to a frame. You will also see how to name a new file and write it after it has been modified. But first, you need to know how to load an existing file into a frame.

Loading Files into “jim”

To load an existing file into a frame in **jim**, do the following:

1. Create a new frame in the upper half of the **jim** window. (If you need help, refer to the previous Steps 5 and 6.)
2. Make the **jim** command line active (inverse video) either by:
 - Moving the arrow cursor inside the command line and clicking button 1

or

 - Entering **Ctrl-j** from the keyboard. This will work only if at least one frame is open.

The command line is now ready to accept commands.

3. In the command line, enter **e filename** (file to be edited) and press **Return**. The arrow cursor will change to a dead mouse until *filename* is completely loaded into the current **jim** frame.

Note: An easy way to create a new frame and load a specific file into it is the **g** (grab) command which is discussed in the "Other jim Commands" section of this chapter.

Selecting Text

In **jim**, the text is edited after it has been selected. Selected text is highlighted in inverse video. Button 1 and the "text cursor" are used to select areas of text. An area of text can be anything from a single character to the entire displayed contents of a frame.

Positioning the "text cursor"

The "text cursor" is the narrow vertical bar that marks text position in a frame (shown in Figure 13-1). All text selection either precedes or follows the "text cursor." You can change the location of the "text cursor" by moving the mouse arrow cursor and clicking button 1. The "text cursor" will move to the null space nearest the arrow cursor when button 1 is clicked. Button 1 can be clicked to move the "text cursor" any number of times without affecting the text.

Selecting Words

1. In the current text frame, position the arrow cursor near the middle of a word and click button 1. The "text cursor" should appear within the word.
2. Without moving the mouse, click button 1 a second time. The word with the "text cursor" within it will be selected (highlighted) in inverse video.

Note: If the mouse is moved before button 1 is clicked the second time, the "text cursor" will change position and the word will not be selected.

Notice that the word is selected up to the next delimiter (space, comma, colon, semicolon, bracket, and so forth). If button 1 is clicked a third time, the word will become normal video (unselected state) again.

Note: Words can also be selected with the "text cursor" at either end of the word.

Selecting a Line

1. Position the "text cursor" at the beginning or ending of a line.
2. Without moving the mouse, click button 1. The entire line will be selected in inverse video.

Selecting an Arbitrary Continuous Text String

1. Position the "text cursor" at the beginning or ending of the text string to be selected.
2. Depress (and hold) button 1. Move the mouse around and notice that as the mouse moves, text is being highlighted in relation to mouse cursor movement.
3. When the desired text is highlighted, release button 1. The text should remain highlighted.

Deleting Text

Text (including blank lines and spaces) can be deleted by using the mouse or the keyboard.

Deleting Text with the Mouse

1. In a current text frame, select the text string to be deleted as previously explained under "Selecting Text."
2. Depress button 2 and highlight the **cut** item from the menu.
3. Release button 2. The highlighted text will be deleted from the frame.

Note: The text from the last deletion is stored in the "save buffer" and can be retrieved if you need it. However, the next deletion rewrites the contents of the "save buffer" and the previously "saved" text can no longer be retrieved. (See "Save Buffer Considerations" of this chapter for more information.)

Deleting Text Using the Keyboard

Position the "text cursor" to the right side of the text to be deleted and press the **Back Space** key. Every time you hit **Back Space**, a character will be deleted.

Adding New Text

Use one of the following methods to add text to a frame:

- Input text directly into an existing or empty frame by typing in the desired text (explained earlier in this chapter).

Note: Typed text always follows the "text cursor."

- Move and copy text within the frame and/or to another frame.
- Use standard input/output redirection. (Discussed in the "I/O Redirection Commands" section of this chapter.)

Moving and Copying Text

Moving and copying text within a frame and/or to another frame is mouse dependent. Button 2 is used to **cut** (remove), **snarf** (copy), and **paste** (add) text in the same frame or between frames. Buttons 1 and 3 are used for cursor positioning and frame selection.

Moving or Copying Text within a Frame

1. In the current frame, select the text to be moved or copied.
2. Depress button 2 and select **cut** or **snarf** from the menu.
3. Release button 2. If **cut** was selected, the highlighted text is removed from the frame and placed in the "save buffer." If **snarf** was selected, the highlighted text is copied into the "save buffer" and the original text remains unchanged.
4. Using button 1, position the "text cursor" to the desired location of text insertion.

5. Depress button 2 and select **paste** from the menu.
6. Release button 2. The text will be inserted at the "text cursor."

Note: The pasted text remains selected and highlighted after the **paste** operation is completed.

Moving or Copying Text to Another Frame

1. Select the text with **cut** or **snarf**.
2. Make the receiving frame current either by positioning the cursor in the receiving frame and clicking button 1 or by selecting the receiving frame from the button 3 menu.

Note: A new frame that has not been named is listed in the button 3 menu either as an asterisk (*) if the frame is not current or by a period(.) if it is current.

3. In the receiving frame, position the "text cursor" to the immediate right of where text is to be inserted.
4. Depress button 2 and select **paste**.
5. Release button 2. The previously selected text will be inserted at the "text cursor."

Note: The pasted text remains selected and highlighted after the **paste** operation is completed.

Changing Text

To quickly change text:

1. Use button 1 to select the text string to be changed, highlighting it in inverse video.
2. Type in the replacing text. The highlighted string will be replaced (an implicit **cut**) as soon as the first character of the replacing text is typed.

The replaced (highlighted) text is stored in the "save buffer." It can be retrieved with a button 2 **paste** command.

Save Buffer Considerations

The **jim** editor has a single save buffer. The save buffer stores selected inverse video text after an editing command action. Each new editing command action overwrites what was previously stored in the save buffer. The **cut**, **snarf**, and “changing text” (previous section) operations store the selected text in the save buffer. The **Esc** key can also store text in the save buffer. (The use of the **Esc** key is explained in the “Special Characters” section of this chapter.)

The **paste** command is used to insert or retrieve the save buffer contents.

If you want to keep the save buffer contents intact while continuing to perform other edit functions, you can either:

- Make a new (empty) frame and then **paste** the save buffer contents into it for retrieval when needed

or

- Use a combination of the **Back Space** key, keyboard keys, and “text cursor” to perform editing functions without disturbing the save buffer contents.

Write Commands

The write command is used to save the contents of a file. Once the contents are saved, you have a permanent record of the file to use later. There are two ways to write a file in **jim**: use mouse button 3 **write** or the keyboard **w** command.

Writing a File Using Button 3

1. Depress button 3 and select **write** on the menu.
2. Release button 3. The cursor will change to a target sight.
3. Move the target sight inside the frame to be written. (Clicking button 1 or button 2 will abort the **write** command selection.)

4. Clicking button 3 will write the file. The following message will appear in the command line: **wrote filename** (*filename* is the name of the file).

Note: If "no filename" appears in the command line, see the "Naming Files" section of this chapter.

Writing Files with the "w" Command

The keyboard **w** command with no argument is used to write a named file. The **w filename** command is used to write the current frame to the specified *filename*. To write files with **w**:

1. Make the desired frame current and then activate the command line.
2. On the keyboard, type **w** and press **Return**. The message **wrote filename** should appear in the command line. If **no file name** appears, the file has not yet been named. If this happens, type **w filename** and press **Return**. The message **wrote filename** appears in the command line.

Naming Files

The keyboard **f** command is used to name, rename, or determine a frame *filename*.

Naming or Renaming a File

1. Make the frame to be named current.
2. Activate the command line.
3. Type **f filename** and press **Return**. *filename* will appear in the command line. The file is now named or renamed.

Note: To make a newly named or renamed file permanent, the file must be written.

Determining the Filename of a Current Frame

1. Activate the command line.
2. Type **f** and press **Return**. The current frame's *filename* will appear in the command line.

String Searches

The **/** command searches forward and the **?** command searches backward. Both will search from the current line for the next occurrence of a predetermined text string.

To perform a string search, do the following steps:

1. In a current text frame, determine a known text string.
2. Activate the command line.
3. Type **/**[*any existing text string*] to search forward or **?**[*any existing text string*] to search backward and press **Return**. The string will be highlighted in inverse video when located.

The frame will be redrawn when necessary. Either string search will wrap around in the frame. When a **/** is used, the search starts immediately following the "text cursor." When a **?** is used, the search starts immediately preceding the "text cursor." If the text string is not found, the following message will appear on the command line:

string not found

The most recent search string is added to the button 2 menu. You can repeat a search by selecting the string in the button 2 menu. Typing a **/** or **?** in the command line without a string will also repeat the most recent string search.

Text strings in **jim** obey the same rules as full regular expressions, as in the UNIX System V command **egrep(1)**. As an example, to search for a *****, you must type **/*** or **?*******.

Other jim Commands

Editor Commands

The active **jim** command line can also execute the following commands:

- E** Edits the file unconditionally.
- q** Conditional quit.
- Q** Unconditional quit.
- =** Displays the text cursor line number location.

Miscellaneous Commands

These commands can be executed from the command line:

- g filename** Initiates the creation of a new frame for editing the named file. The cursor in the **jim** window will change to a sweep cursor (arrow inside a square) which is an indication to sweep a new frame. Button 3 is used to sweep the new frame. (If a frame already exists for the named file, the **g** (grab) command makes that frame current.)
- cd dir** Sets the working directory to *dir*. There is no CDPATH search, but \$HOME is the default *dir*.

I/O Redirection Commands

An active **jim** command line can execute the following UNIX System commands:

- >** Sends the selected text to the standard input of a command.
- <** Replaces the selected text by the standard output of a command.
- |** Replaces the selected text by the standard output of a command, given the original selected text as standard input.

Note: If any of these commands are preceded by an asterisk (*), the command is applied to the entire file, not just the selected text.

As an example, typing `> pwd` in the active command line will cause the working directory to be printed. As a second example, typing `<ls` in the active command line will cause the listing of the current working directory to be inserted at the current cursor position of the last active frame.

The most recent I/O redirection command is added to the button 2 menu for easy repetition.

Frame Positioning and Scrolling

Positioning in a current text frame can be done from the command frame or with the mouse. Scrolling can only be done with the mouse.

Positioning Cursor from the Active Command Line

Type any existing line number and press **Return**. The text of the line number typed will be highlighted in inverse video. The frame will be redrawn if necessary.

Using the Mouse to Position the Cursor

1. Move the mouse arrow to a place on the current frame.
2. Press button 1 on the mouse. The cursor will move to the point immediately preceding the mouse arrow.

Scrolling within a Frame

Notice the small “tick” inside the current text frame’s inverse video “scroll bar.” The tick indicates the relative position of the file displayed inside the frame. By moving the tick inside the scroll bar, you can scroll text within a frame. To position the tick, move the mouse arrow up or down in the scroll bar area and press a mouse button, as determined from the following:

- Button 1 moves the line at the top of the frame to the location where button 1 is clicked within the scroll bar. This causes the file to scroll backward until the first line of the file is at the top of the frame.

- Button 2 moves the frame file absolute position (indicated by the “tick”) to where button 2 is clicked. If button 2 is clicked in the middle of the scroll bar, the middle portion of the file will be displayed. When clicked at the end, beginning, or any fraction thereof, that portion of the file will appear in the frame. This feature is useful for scrolling in long files.
- Button 3 moves the line of text where button 3 is clicked within the scroll bar to the top of the frame. This causes the file to scroll forward.

Note: Buttons 1 and 3 can be held down to repeat.

Special Characters

The following characters cannot be inserted in the text of a **jim** frame:

- **Back Space (Ctrl-h)** erases characters to the left of the cursor.
- **Ctrl-w** erases back to the word boundary preceding the selected text or “text cursor.”
- **Esc** key selects the text typed since the last mouse button was clicked. If an **Esc** key is pressed twice immediately after typing text, it is identical to a **cut** and the text will be stored in the save buffer. After pressing **Esc** twice, the button 2 **paste** command can then be used to “undo” changes.

Button 3 Menu Features

Mouse button 3 has four commands in its menu: **new**, **reshape**, **close**, and **write** which provide control of the selected frame.

As you have already seen, the **new** command is used to create frames inside the **jim** window.

Note: When the sweep cursor appears after selecting **new**, clicking button 3 causes the new frame to be automatically drawn to the size of the entire **jim** window.

Jim Operations

The **reshape** command changes the shape of a frame, as follows:

1. Press button 3 and select **reshape**.
2. Release button 3. The cursor is now a target sight.
3. Move the target sight cursor over the frame to be reshaped.
4. Click button 3. The "sweep cursor" (square cursor with an arrow) appears as the cursor. Now, you have a choice:
 - Click button 3 again and the selected frame will automatically reshape itself to fill the **jim** window.

or

- Move the sweep cursor to locate a corner of the frame being reshaped. Depress button 3 and move the cursor to reshape the frame. The frame will be defined and loaded with the file when button 3 is released.

The **close** command removes a selected frame from the **jim** window. The file still exists as a UNIX System file; only the associated frame is shut down. To **close** a frame:

1. Depress button 3 and select **close** on the menu.
2. Release button 3. The cursor is now a target sight.
3. Move the target sight inside the frame to be closed.
4. Click button 3. The frame disappears. If you click button 1 or button 2, the **close** command is canceled.

Note: The message *filename changed* will appear in the command line if changes were made to the file and not written. A second **close** selection will succeed.

The rest of the button 3 menu lists frame *filenames* that are available for editing. To work in a different file, select the *filename* from the menu. If the frame is already open, it is simply made the current frame. If the file is not open on the screen, the cursor will switch to a sweep cursor to prompt for a rectangle to be swept out with button 3.

The *filename* line may include:

- An apostrophe, indicating the file (frame) has been modified since last written
- An asterisk or a period, indicating the frame is open (asterisk) or the current frame (period)
- A blank, indicating a frame with no name

Note: The *filename* may be abbreviated, but the last component is always complete.

Note: If the entire **jim** window is moved or reshaped, all open text frames will be closed. The text frames can be reopened by selecting the filenames in **jim**'s button 3 menu.

Quitting Jim

The **q** command is used to quit the **jim** editor. Typing **q** in the command line may cause the **files changed** message to appear (this is a reminder that a file[s] has been modified and a write should be done). Typing a second **q** will cause **jim** to exit and return to a UNIX System V shell. The capital **Q** command ignores modifications and exits **jim** immediately. See the **JIM(1)** manual page in the *630 MTG Software Reference Manual* for detailed information.

Command Line Diagnostic Messages

Figure 13-2 gives a list of diagnostic messages that appear in the *jim* command line. An explanation of each message is also included.

Diagnostic Message	Explanation
bad directory	You entered an incorrect path name or you do not have permission to access the named directory. Use the correct path name and check directory access permissions.
can't open <i>filename</i>	You are trying to access an illegal file or you do not have read permission for the named file. Make sure <i>filename</i> is accessible.
file already exists	You tried to write a file that already exists in the current working directory. Enter write command again to overwrite the file.
files changed	File changes were made but not written. Either write the file or enter q again. A second q or Q command ignores previous file changes and quits immediately.

Figure 13-2: Diagnostic Messages (Sheet 1 of 4)

Diagnostic Message	Explanation
file modified since last read/written	An attempt was made to write a file that was changed in another frame. To avoid confusion, close the frame you don't want. Then, you can write the other frame without receiving a diagnostic message.
no file name	You tried to write a file that does not have a <i>filename</i> assigned. Give the frame a <i>filename</i> by using the f or w command.
--RE error: operand expected for *, + or ?	An attempt was made to search for a string that has the characters *, +, or ? in it. Precede these characters with a back slash ("\<"). If one of these characters is at the end of the string, the character will be ignored and this message will not appear.
sorry; can't edit huge selection or sorry; first deselect that huge thing	Selected text is beyond the area of the frame. Redraw the frame, making it larger.

Figure 13-2: Diagnostic Messages (Sheet 2 of 4)

Diagnostic Message	Explanation
<i>string not found</i>	The text string searched for was not found. Check <i>string</i> to make sure you typed it correctly. If <i>string</i> was typed correctly, an invisible character (i.e., control characters) may be part of the string, or the string does not exist in the file. Control characters cannot be searched for in jim .
syntax	An illegal command was entered. This usually happens when ed instead of e is used to open a file for editing. Enter command with correct syntax.
too many files open	Message is displayed when too many files are open. Reduce the number of opened frames.
UNIX message unknown?	A message was written to your jim window. No action is required but you may want to make another window active and investigate the message.

Figure 13-2: Diagnostic Messages (Sheet 3 of 4)

Diagnostic Message	Explanation
warning: <i>filename</i> already loaded	Message is displayed if an opened file is loaded into a second jim frame. No action is required but you may want to close one of the frames.
last char not newline; <i>wrote filename</i>	You tried to write a file but the last character of the file is not a new line. In the frame, make sure the text cursor is on the last blank line by pressing the Return key at end of last file line; then write the file again.
wrote <i>filename</i>	The file in the current frame was written as requested. No action is required.
you typed: <i>repeat of your typed input</i>	The typed input is not acceptable. Enter an acceptable command.

Figure 13-2: Diagnostic Messages (Sheet 4 of 4)

Interactively Recovering Lost Files

If **jim** is exited abnormally (loss of power, etc.), the 630 MTG will attempt to create a recovery file for each window where **jim** was downloaded. The recovery file(s) will be located in the \$HOME directory. The name of the recovery file will be in the form:

jim.string

where *string* is a unique alphanumeric code for each **jim** window. Mail will be sent to your *login id* stating that the files were saved and specifying the executable file name.

The recovery file is then used interactively to recover the files that had been modified but not written when the **jim** window was removed.

To restore files to your \$HOME directory, type in the following:

jim.string

To restore files to another directory, **cd** to that directory and type in the following:

\$HOME/jim.string

The recovery program causes each modified file to be listed individually in the following manner:

file (modified)?

You can choose whether or not to restore each file by entering **y** (yes) or **n** (no).

Files that have not been named will be listed as "**nameless_1**, **nameless_2**, etc.

Caution: Anytime a file is being recovered, the "recovered" file overwrites the contents of another file with the same *filename*. Also, if **jim** has been downloaded into more than one window, the "nameless" files from one window will have the same *filename* (**nameless_1**, **nameless_2**, etc.) as those from the other **jim** window(s). There are a couple of ways to avoid overwriting these files during recovery. Recover the nameless files from the first window and then rename them before recovering the files from the other window(s). Or, recover the nameless files in different directories.

See the **JIM(1)** manual page in the *630 MTG Software Reference Manual* for more information on recovering files.

Keyboard Command Summary

The following list is a summary of the **jim** keyboard commands and their functions that are used in this chapter.

- e filename** Used to load *filename* into a **jim** frame for editing.
- E filename** Edits unconditionally; loads *filename* regardless of the content of the current frame.
- f filename** Used to name or rename the current frame.
- g filename** Used to create a new frame and load *filename* for editing (grab).
- q** Is a conditional quit to **jim**; if all modified frames have been written.
- Q** Is an unconditional quit; ignores all modifications.
- w** With no argument is used to write a named file. **w filename** writes the current frame to the specified *filename*.
- cd dir** Sets working directory to *dir*.

Control Characters

- Ctrl-h** (**Back Space**) erases characters to the left of the cursor.
- Ctrl-j** Toggles between activating the command line frame to receive keyboard input and activating the last active frame.
- Ctrl-w** Erases back to the word boundary preceding the selected text or "text cursor."

Escape Character

- Esc** Used to select, cut, and pastes text that has been typed since the last mouse button was clicked.

Symbol Characters

- = Displays the text cursor line number for the current frame.
- / *string* Performs a forward search for the next occurrence of *string*.
/ without *string* will repeat the last *string* search.
- ? *string* Performs a backward search for the next occurrence of *string*.
? without *string* will repeat the last *string* search.
- > Sends selected text to the standard input of a command.
- < Replaces selected text by the standard output of a command.
- | Replaces the selected text by the standard output of a command,
given the original selected text as standard input.
- (*line #*) Positions the current frame to the specified *line number*.

Chapter 14: Icon Editing

Introduction	14-1
Bitmaps and Texture16s	14-2
How Icons Are Stored	14-2
Using Icon	14-3
Initiating an Icon Session	14-3
Selecting Bitmap or Texture16	14-4
The Icon Menu	14-5
Drawing With Icon	14-9
Move Icon	14-9
Copy Icon	14-10
Invert	14-10
Erase Icon	14-10
Flip Icon	14-11
Shear Icon	14-11
Stretch Icon	14-12
Duplicate Icon	14-12
Read Icon File	14-12
Background Grid	14-13
Change Mouse Cursor	14-13
Bitblt Operator	14-13
Write Icon Files	14-15
Quit Icon	14-15
Using Icons in Programs	14-15
Using Icons in Bitmaps	14-15
Using Icons for Background Textures and Mouse Cursors	14-17
Supplied Icons	14-18

Introduction

icon is a visual editor for graphical images commonly referred to as "icons". Icons are used as image data for Bitmaps and Texture16s which are ultimately incorporated into application programs to enhance user interface. See the "Graphics Environment" Chapter for details on the Bitmap and Texture16 graphical data types.

Before going any further, a few definitions are necessary:

- | | |
|-------------|--|
| icon | This is the graphical image editor for the 630 MTG. It is shown in lower case, bold type. |
| icon | This is a pattern drawn in the grid of the icon editor. |
| Texture16 | This is a 630 MTG graphical data type. The size of this data type is always 16 by 16 bits. |
| Bitmap | This is a 630 MTG graphical data type. There is no size restriction on the image data of a Bitmap. |

After gaining familiarity with the **icon** operations presented in this chapter, you can use the **ICON(1)** manual page in the *630 MTG Software Reference Manual* as a reference.

Bitmaps and Texture16s

Icons created with the **icon** editor can be used in two different 630 MTG graphical data types: Bitmaps and Texture16s. In the case of Bitmaps, the icon is used as the Bitmap's image data. This is the array of data pointed to by **base** in a Bitmap data structure. The Bitmap image data has no size restrictions. Texture16s, on the other hand, require an icon that is 16 by 16 bits. Texture16 data structures can be used as background textures and mouse cursors.

How Icons Are Stored

Depending on whether you are creating image data for a Bitmap or Texture16, the **icon** editor will store the icon differently. The image data for a Bitmap will be stored in a file as a list of 32-bit hexadecimal quantities. The file will not include any of the header and trailer information to declare the image data as a Bitmap. This allows you to combine the files to create larger Bitmaps.

When an icon is stored as a Texture16, the appropriate Texture16 header and trailer information is added. You do not have to add any additional information to use the Texture16 data structure in an application.

Using Icon

Initiating an Icon Session

To download the **icon** program, create a new window and type in: **icon**. The default display consists of a large grid representing 50 by 50 pixels in the lower right-hand corner of the window. Each square in the grid represents one pixel on the 630 MTG display.

The default size of the grid may be changed by using the "-x" and "-y" options. For example, to display a 42 by 34 pixel grid, type:

```
icon -x 42 -y 34
```

While the cursor is positioned over the grid, and the **icon** window is current, pressing mouse button 1 draws a pixel, and pressing button 2 undraws a pixel. In the upper left-hand corner of the display, the icon you are drawing will appear in actual size. Figure 14-1 shows the **icon** window with the supplied icon "banana" drawn in the grid. Icons supplied with the 630 MTG Software Development Package can be found in the subdirectories located under *\$DMD/Icons*.

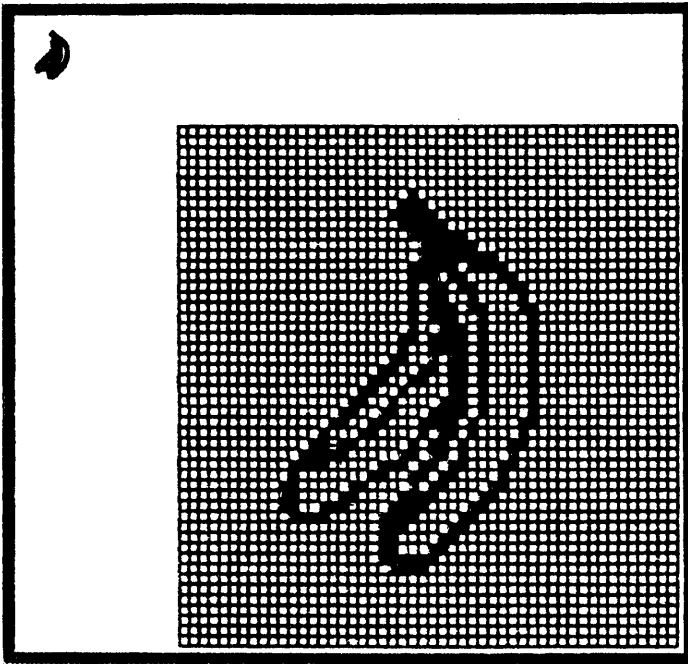


Figure 14-1: The **icon** Editor Display

Selecting Bitmap or Texture16

The **icon** editor reads, writes, or changes icons for both Bitmap image data and Texture16 data structures. You select the icon type by pressing mouse button 2 (for a Texture16 data structure) or button 3 (for Bitmap image data) after selecting a write, read, or change command from the **icon** editor's menu. For example, when you have drawn an icon and select the write command, **icon** requests the type of icon you want written by changing the mouse cursor to the "Sweep Rect or Get 16 x 16 Box" cursor.

Using Icon

3. Every icon that appears in the menu is shown in the **help** message with a short description.
4. Click any mouse button to remove the **help** message.

The following gives a short description of each command in order from left to right, top to bottom (as shown in the menu):

MENU

DESCRIPTION



Arrow: Moves the icon to another portion of the grid.



Copy: Copies the icon to another portion of the grid.



Invert: Inverts (bitwise complements) an icon.



Erase: Erases the icon.



Reflect X: Flips the icon on the x axis.



Reflect Y: Flips the icon on the y axis.



Rotate +: Rotates the icon clockwise 90 degrees.



Rotate -: Rotates the icon counterclockwise 90 degrees.

MENU

DESCRIPTION



Shear X: Shears the icon along the x axis. (For example, a rectangle will change to a parallelogram shifted along the x axis.)



Shear Y: Shears the icon along the y axis. (For example, a rectangle will change to a parallelogram shifted along the y axis.)



Stretch: Stretches (or shrinks) the icon.



Duplicate: Duplicates an icon over a larger (or smaller) area of the grid.



Read File: Reads a file containing Bitmap image data or a Texture16 data structure.



Background Grid: Draws a reference grid divided into 16 by 16 squares with borders highlighted.



Pick Cursor Icon: Changes the mouse cursor to a Texture16 selected in the grid.

Using Icon

MENU

DESCRIPTION



Write File: Writes the icon to a file as Bitmap image data or a Texture16 data structure.



Bitblt: The bitblt operator allows you to alter the source and destination areas specified on the grid.



Help: Prints the **icon** help message. Press any mouse button to continue.



Exit: Exits the **icon** program. Confirm request with a button 3 press.

The following icons appear as mouse cursors after selecting one of the icons from the command menu:

ICON

DESCRIPTION



Wait: Wait while **icon** requests I/O.



Mouse Inactive: The mouse is inactive; wait.



Menu on Button 3: Press button 3 to display menu.

ICON**DESCRIPTION**

Sweep Rect: A prompt to sweep a rectangle (button 3).



Sweep Rect or Get 16 x 16 Box: Select Texture16 or Bitmap (button 2 or 3). Button 2 defines a Texture16 by displaying a large 16 by 16 "box". Button 3 defines a Bitmap by allowing you to sweep a rectangle.

Drawing With Icon

Generally, you can create icons by pointing to the **icon** grid with the mouse cursor. The bits pointed to by the mouse cursor are set by clicking button 1 and cleared by clicking button 2. You may draw (or undraw) a bit at a time by clicking the button; or, by holding the button down, draw (or undraw) many bits while moving the cursor.

Note: In the following procedures, you will be instructed to select a Texture16 or Bitmap. For more information, refer to the previous section "Selecting Bitmap or Texture16" in this chapter.

Move Icon

An icon is moved within the **icon** grid by using the **arrow** menu selection as follows:

1. Select the **arrow** from the **icon** command menu.
2. In response to the "Sweep Rect or Get 16 x 16 Box" cursor, select an area of the grid as a Texture16 or Bitmap image data (see "Selecting Bitmap or Texture16").
3. Move the mouse cursor, now a "large box", over the area of the grid where you want the icon to appear; then click button 2 or 3.
4. The icon is drawn into the new location on the grid.

Copy Icon

After you have drawn an icon, you may copy it to another portion of the grid using the **icon copy** command as follows:

1. Select the **copy** command from the **icon** command menu.
2. In response to the "Sweep Rect or Get 16 x 16 Box" cursor, select a Texture16 or Bitmap.
3. Move the cursor, now a "large box", to the portion of the grid where you want the copy to appear and click button 2 or 3.
4. The copy appears in the desired area. Note that copies are clipped to the grid area.

Note: You can copy the icon as many times as you wish. However, the icon may become garbled and unrecognizable as copies overlap. Watch the actual size representation of the icon in the upper left-hand corner of the **icon** window. This is a true indication of the icon in the grid.

Invert

To invert (bitwise complement) a section of the **icon** grid:

1. Select the **invert** from the **icon** command menu.
2. In response to the "Sweep Rect or Get 16 x 16 Box" cursor, select a Texture16 or Bitmap.
3. When button 2 (or 3) is released, the selected area is inverted.

Erase Icon

To clear an area of the **icon** grid:

1. Select the **erase** from the **icon** command menu.
2. In response to the "Sweep Rect or Get 16 x 16 Box" cursor, select a Texture16 or Bitmap.
3. When button 2 (or 3) is released, the area selected is deleted.

Flip Icon

The **reflect X**, **reflect Y**, **rotate -**, and **rotate +** commands flip the icon in the indicated direction. The procedure is the same for each operation:

1. Select the desired command from the **icon** command menu.
2. In response to the "Sweep Rect or Get 16 x 16 Box" cursor, select a Texture16 or Bitmap.
3. At the release of mouse button 2 or button 3, the icon flips.

Shear Icon

The **shear x** command shears an icon along the horizontal axis. To shear the icon:

1. Select the **shear x** icon from the menu.
2. In response to the "Sweep Rect or Get 16 x 16 Box" cursor, select a Texture16 or Bitmap. The cursor changes to a "small box".
3. Move the "small box" left or right.
4. Click button 3. The **shear x** command causes the y-axis to tilt in the direction of the "small box". The icon tilts (or skews) in the direction and distance determined by the direction and distance of the "small box" relative to the closest corner of the Texture16 or Bitmap.

The **shear y** command shears an icon along the vertical axis. To shear the icon:

1. Select the **shear y** icon from the menu.
2. In response to the "Sweep Rect or Get 16 x 16 Box" cursor, select a Texture16 or Bitmap. The cursor changes to a "small box".
3. Move the "small box" up or down.
4. Click button 3. The **shear y** command causes the x-axis to tilt in the direction of the "small box". The icon tilts (or skews) in the direction and distance determined by the direction and distance of the "small box" relative to the closest corner of the Texture16 or Bitmap.

Stretch Icon

The **stretch** command stretches (or shrinks) the icon. To change icon size:

1. Select the **stretch** icon from the menu.
2. In response to the "Sweep Rect or Get 16 x 16 Box" cursor, select a Texture16 or Bitmap.
3. Fix one corner of the new icon by pressing and holding button 3.
4. Move the cursor to the diagonally opposite corner of the new icon.
5. Release the button that you pressed. The icon changes to fill the "box".

Duplicate Icon

The **duplicate** command duplicates an icon over a larger (or smaller) area within the grid. To duplicate a texture:

1. Select the **duplicate** from the **icon** command menu.
2. In response to the "Sweep Rect or Get 16 x 16 Box" cursor, select a Texture16 or Bitmap.
3. Using either button 2 or button 3, define the area where the icon is to be duplicated as a Texture16 or a Bitmap.
4. The icon is repeatedly duplicated until it fills the new area.

Read Icon File

To read an existing Bitmap image or Texture16 data structure stored in a file, do the following:

1. Select the **read file** command from the **icon** menu. You are required to type in the name of the file to be read.
2. If the file contains a Texture16 data structure, the mouse cursor changes to a 16 by 16 box, defining the area the Texture16 covers on the grid. Move the "box" to the position on the grid where you want the Texture16 to appear.
3. Click button 2. The Texture16 is drawn in the grid.

4. If the file contains Bitmap image data, the cursor changes to a "box" enclosing the area of the grid covered by the Bitmap. Move the "box" to the position where you want the Bitmap to appear.
5. Click button 3. The Bitmap is drawn in the grid.

Note: Files to be read by **icon** that have been modified or manually created must correspond precisely to the format used by **icon** when it writes a file.

Background Grid

The **icon** grid can be divided into 16 by 16 pixel areas by selecting the **background grid** command from the menu. Selecting the **background grid** command again removes it. These 16 by 16 areas are useful when drawing Texture16s or to determine how many 16 by 16 pixel areas an icon occupies.

Change Mouse Cursor

You can change the mouse cursor to display the Texture16 you are developing by doing the following:

1. Select the **pick cursor icon** from the **icon** command menu. The cursor changes to a 16 by 16 pixel "box".
2. Move the "box" over the area you want, then click button 2. The cursor changes to the area you selected. This lets you test the Texture16 for its suitability as a mouse cursor.
3. Clicking any mouse button cancels the command and returns the cursor to its previous shape.

Bitblt Operator

The **bitblt** command is used to alter the source and destination areas specified on the grid. You can perform bitwise copies and moves on textures in the grid.

To perform a bitwise move:

1. Select the **bitblt** command from the **icon** command menu. The cursor changes to the "menu on button 3" icon.

Using Icon

2. Press and hold button 3. A menu is displayed showing:

src := src
src := 0

where: **src := src** copies icons and **src := 0** moves icons.

3. Select **src := src** or **src := 0**.
4. The cursor changes to "menu on button 3" icon.
5. Press and hold button 3. A menu is displayed showing:

dst := src
dst := src or dst
dst := src xor dst
dst := 0

where:

dst := src

copies (or moves) the selected icon to another portion of the grid. If the new icon overlaps the original icon, the original is erased.

dst := src or dst

copies (or moves) the selected icon to another portion of the grid. If the new icon overlaps the original icon, the bits are or'ed.

dst := src xor dst

copies (or moves) the original icon to another portion of the grid. If the new icon overlaps the original icon, the bits are exclusive or'ed.

dst := 0

erases the icon.

6. Select an entry from the menu. The mouse cursor changes to the "Sweep Rect or Get 16 x 16 Box" icon.

7. Select a **Bitmap** or **Texture16**.
8. Move the cursor to the area in the grid you want the texture to appear and click button 2 or 3.
9. The icon is moved (or copied) to the new location in the grid defined by the menu selection.

Write Icon Files

To write an icon from the **icon** grid to a file:

1. Select the **write file** from the **icon** command menu.
2. Select a **Bitmap** or **Texture16**.
3. Type in a name for the file. While writing the file, the **icon** window changes to reverse video.

Files are saved in the window's current directory unless a full path name is specified.

Quit Icon

To exit the **icon** editor:

1. Select **exit** from the **icon** command menu. The cursor changes to a "smoking gun".
2. Click button 3 to exit the **icon** editor. Click button 1 or 2 to cancel the exit request.

Using Icons in Programs

After saving an icon in a file as a **Texture16** or as **Bitmap** image data, it can be incorporated into an application program. The following sections discuss how to use the saved icons in an application.

Using Icons in Bitmaps

When the **icon** editor writes an icon to a file as **Bitmap** image data, the file contains a comma-separated list of 32-bit hexadecimal values. There may be one or more 32-bit values on each line depending on the width of the icon that was written to the file.

The image data written by **icon** must be edited in order to use it as the image data for a Bitmap. To do this, you must declare the list of 32-bit quantities as an array of unsigned longs and then initialize **base** in the Bitmap data structure to point to this array. (Be sure to typecast the address as a pointer to a **Word**.) The **width** and **rect** fields of the Bitmap must be initialized according to the image data size. See the "Graphics Environment" Chapter for more details on the Bitmap data type.

The following example program shows the results of editing a file output by **icon** to incorporate it into a program as the image data for a Bitmap.

```
#include <dmd.h>

unsigned long bits[] = {           /* Header line - defines */
                                   /* the icon data as an array */
                                   /* of unsigned longs.          */

    0x0001FFFE,                    /* Start of the data file */
    0x00021086,                    /* that was saved by the */
    0x0004210A,                    /* icon program for the */
    0x000FFFF2,                    /* icon named "cube".    */
    0x00108432,
    0x00210852,
    0x007FFF96,
    0x0084219A,
    0x01084292,
    0x03FFFCB2,
    0x021084D2,
    0x02108496,
    0x0210859A,
    0x02108692,
    0x03FFFCB2,
    0x021084D2,
    0x02108494,
    0x02108598,
    0x02108690,
    0x03FFFCB2,
    0x021084C0,
    0x02108480,
    0x02108500,
    0x02108600,
    0x03FFFC00,
    0x00000000
```

```

};                                /* trailer for icon bitmap */

    /* beginning of sample program */

Point add();

Bitmap example = {
    (Word *)bits, /* pointer to data */
    2,           /* width in Words of data area */
    0,0,32,26,  /* rectangle */
    0           /* unused, always zero */
};

    /* This program draws the icon bitmap. */
    /* It makes the icon "walk" down the window */
main()
{
    int i;
    Point j;

    for (i=10; i<1000; i+=40)
    {
        j = add(Pt(i,i), Direct.origin);
        sleep(15);
        bitblt (&example, example.rect, &display,
                j ,F_STORE);
    }
    request(KBD);
    wait(KBD);
}

```

See the "Graphics Environment" Chapter in this document and the manual pages **GLOBALS(3R)**, **STRUCTURES(3R)**, **BITBLT(3R)**, and **RESOURCES(3R)** in the *630 MTG Software Reference Manual* for details on how this sample program works.

Using Icons for Background Textures and Mouse Cursors

The Texture16 data structures written by **icon** can be used in two different ways: background textures and mouse cursors. The graphics routine **texture** is used to display background textures. See the manual page **TEXTURE(3R)** in the *630 MTG Software Reference Manual* for an example.

Using Icon

An application can call the mouse cursor control routine **cursswitch** to switch the standard mouse cursor to any Texture16 data structure. See the manual page **CURSOR(3R)** in the *630 MTG Software Reference Manual* for an example of how to use the **cursswitch** routine with **Texture16** data structures.

Supplied Icons

Icons supplied for the 630 MTG are provided under the directory *\$DMD/icons*. To display these icons, download the **icon** program and read the icon files into it.

Chapter 15: C Compilation System

Introduction	15-1
The 630 MTG Compiler	15-3
Compiler Options	15-4
Other Utilities	15-5
mc68ar	15-5
mc68conv	15-6
mc68cprs	15-7
mc68dis	15-7
mc68dump	15-8
mc68ld	15-9
mc68lorder	15-9
mc68nm	15-10
mc68size	15-11
mc68strip	15-11
Register Use	15-12
C Language	15-16

Introduction

This chapter describes the 630 MTG C Compilation System (CCS) Compiler. See the "MC68000 UNIX System Assembler" Chapter for more details on the CCS.

The CCS is a package of tools used to create and test programs for the 630 MTG. These tools allow high-level program coding and source-level testing of code. Within the CCS, a C language compiler converts C language programs into assembly language programs that are ultimately translated into object files by the **mc68as** assembler. The **mc68ld** link editor collects and merges object files into executable loads. Each of these tools preserves all symbolic information necessary for meaningful symbolic testing at the C language source level. In addition, a utility package aids in testing and debugging.

The default output file from the C language compiler is an object file named **dmda.out**. It has a format called the Common Object File Format (COFF). The object file is executable in the 630 MTG if no errors or unresolved references are found. The file contains a header with size information, program sections, and a symbol table. Each section is composed of a section header, data, and relocation and line number information. Depending on the assembler or link editor options used to produce the object file, the file may be devoid of relocation entries, line number entries, the symbol table, or compiler-generated symbols.

The CCS provides a variety of utilities to read and manipulate object files. Among the functions performed by the utilities are listing, reducing, or deleting various parts of an object file or symbol table.

This chapter gives an overview of the CCS utilities. The most commonly used of these utilities is the 630 MTG Compiler **dmdcc** which is described in the following section. The other CCS utilities are described in the "Other Utilities" section of this chapter. The CCS utilities are:

- dmdcc** Compiles programs for the 630 MTG. Invokes the compiler, the assembler (**mc68as**), and the link editor (**mc68ld**) as appropriate.
- mc68ar** Creates and updates an archive.

Introduction

- mc68as** Assembles MC68000* microprocessor source code files. See the "MC68000 UNIX System Assembler" Chapter for a discussion of **mc68as**.
- mc68conv** Converts MC68000 microprocessor object files from one host machine format to another host machine format.
- mc68cprs** Compresses object files by removing duplicate structure and union descriptors.
- mc68dis** Disassembles object files to allow assembly level debugging.
- mc68dump** Dumps selected parts of the named object files.
- mc68ld** Links together object files for execution.
- mc68lorder** Generates an ordered listing of object files suitable for link editing in one pass, as done by **mc68ld**.
- mc68nm** Prints the symbol table for an object file.
- mc68size** Reports the number of bytes of text, uninitialized data, and initialized data (and their sum) included in an object module.
- mc68strip** Reduces file storage overhead by removing symbolic testing information from an object file.

For more information on the CCS utilities, see the associated manual page in the *630 MTG Software Reference Manual*.

* Trademark of Motorola, Inc.

The 630 MTG Compiler

The main command of the CCS is **dmdcc**; it operates much like the UNIX System **cc** command. To use the compiler, first create a file (typically by using any UNIX System text editor) containing C language source code. The name of the file created must have a special format; the last two characters of the file name must be *“.c”* — as in, *file.c*.

Next, enter the CCS command

```
dmdcc file.c
```

to invoke the compiler on the C language source file *file.c*. The compilation process creates a binary file named **dmda.out** that reflects the contents of *file.c* and any referenced library routines. The resulting binary file, **dmda.out**, can then be downloaded to the 630 MTG.

Options can control the steps in the compilation process. When none of the controlling options are used and only one file is named, **dmdcc** automatically calls the assembler, **mc68as**, and the link editor, **mc68ld**, thus resulting in a downloadable file named **dmda.out**. If more than one file is named in a command,

```
dmdcc file1.c file2.c file3.c
```

then the output will be placed in files *file1.o*, *file2.o*, *file3.o*, and the downloadable file, **dmda.out**.

The **dmdcc** compiler also accepts input file names with the last two characters **.s**. The **.s** signifies a source file in assembly language. If the **-c** option is specified, the **dmdcc** compiler passes this type of file directly to **mc68as**, which assembles the file and places the output in a file of the same name with **.o** substituted for **.s**. Otherwise, **dmdcc** calls **mc68ld** and creates **dmda.out**.

dmdcc is based on a portable C language compiler and translates C language source files into MC68000 assembly language. Whenever **dmdcc** is used, the **mc68cpp** C language preprocessor is called. The preprocessor performs file inclusion and macro substitution. The preprocessor is always invoked by **dmdcc** and need not be called directly by the programmer. The expanded files are translated from C language to MC68000 assembly language. Then, unless the appropriate flags are set, **dmdcc** calls the assembler and the link editor to produce an executable file.

Compiler Options

All options recognized by the **dmdcc** command are listed in the **DMDCC(1)** manual page of the *630 MTG Software Reference Manual*.

By using the appropriate options, compilation can be terminated early to produce one of several intermediate translations, such as:

- Relocatable object files (**-c** option)
- Assembly source expansions for C language code (**-S** option)
- Output of the preprocessor (**-P** option).

In general, the intermediate files may be saved and later resubmitted to the **dmdcc** command, with other files or libraries included as necessary.

When compiling C language source files, the most common practice is to use the **-c** option to save relocatable files. Subsequent changes to one file do not then require that the others be recompiled. A separate call to **dmdcc** without the **-c** option then creates the linked **dmda.out** file. A relocatable object file created under the **-c** option is named by adding a **.o** suffix to the source file name.

Other Utilities

mc68ar

The **mc68ar** utility is used to maintain groups of files that are part of a single archive file. It is mainly used to create and update library files that are used by the link editor (**mc68ld** command), but it can be used for any similar purpose. When **mc68ar** creates an archive, the archive file is put into a format with headers that are portable across all computers. These headers are placed at the beginning of each archive.

The header is followed by an archive symbol table which is included in each archive that has common object files. This symbol table is automatically created by **mc68ar**. The archive symbol table is used by the link editor to determine what archive members must be loaded during the link edit process. The archive symbol table is rebuilt each time the **mc68ar** command is used to create or update the contents of an archive.

The archive symbol table is followed by the archive file members. A file member header precedes each file member. The file member header format is described in the UNIX System V **ar(4)** manual page.

All the information in the archive header, the archive symbol table, and the archive file member headers is stored in a machine (computer) independent fashion. Archive members can be extracted on any machine, although some members may be host computer dependent and require conversion due to byte ordering differences on different machines.

Suppose you have the following files that you want to archive:

- cars
- cities
- people
- states
- streets

To archive these files, enter the following command:

```
mc68ar q archive1 cars cities people states streets
```

Other Utilities

You should receive the following response:

```
mc68ar: creating archive1
```

To see what is in the archive you just created, enter the following command:

```
mc68ar t archive1
```

You should receive the following response:

```
cars  
cities  
people  
states  
streets
```

mc68conv

Whenever a *COFF* file is moved from one machine to another of different architecture, **mc68conv** should be used to format the resulting file.

Differences in byte ordering and data formats cause object file formats to differ in their symbolic information when produced on machines of disparate architectures. **mc68conv** converts a MC68000 microprocessor object file (for example, **dmda.out**) from the internal format of one machine architecture to that of another architecture. For example, to move a MC68000 microprocessor object file produced on an AT&T 3B20S minicomputer to a VAX*-11/780 minicomputer, one must use **mc68conv** or the resulting file will not be in a usable format.

File conversion is necessary and effective between machines of the following three architectures:

1. A DEC*-style byte ordering with 16-bit word length (for example, PDP*-11/70 Computer)

* Registered trademark of Digital Equipment Corporation

2. A DEC-style byte ordering with 32-bit word length (for example, VAX-11/780 Computer)
3. An IBM*-style byte ordering with 32-bit word length (for example, AT&T 3B20 Computer).

The output of **mc68conv** is a file having the same name as the input file with the suffix of **.v**. Output cannot be redirected from the **mc68conv** command.

mc68conv is best used within a procedure for sending object files from one machine to another. Attempting to convert a file when no conversion is necessary results in an error message, although the input file is copied to the output file.

mc68cprs

The utility **mc68cprs** reduces the size of a MC68000 microprocessor object file by removing duplicate structure and union descriptors. For example:

```
mc68cprs dmda.out sma.out
```

will take **dmda.out** and produce an equivalent file called **sma.out** that will have all duplicate structure and union descriptors removed.

mc68dis

The MC68000 microprocessor disassembler, **mc68dis**, produces an assembly language listing of each input object file. The listing has a two-column format with assembly language statements in the right column and the corresponding binary object code and machine address of the code in the left column.

The disassembler produces a likeness of the assembly language file that was assembled to produce a given object file. Note that the assembly language file produced by **mc68dis** is not the same file that was accepted by the assembler, **mc68as**. **mc68dis** provides a convenient method to obtain a MC68000 microprocessor assembly language listing of C language source programs.

* Registered trademark of International Business Machines Corporation

Three features of the **mc68dis** listing are:

1. The disassembler prints line numbers for each C source line where a breakpoint can be set in square brackets (for example, “[5]” shows the fifth line where execution can be halted for debugging). The line numbers appear in the first column, at the instruction corresponding to the line where a breakpoint can be inserted.
2. The disassembler prints C function names followed by parentheses (for example, “printf()” for the function **printf**). The function names appear in the first column, one line above the instruction that begins the function.
3. The disassembler prints computed addresses within a section when control is to be transferred to those addresses. They are printed within triangular brackets (for example, “<40>” is computed address 40). These addresses appear in the operand field of control transfer instructions following a relative displacement. The computed address is the sum of the relative displacement and the address of the instruction currently being disassembled.

Note that items 1 and 2 occur only if the information exists in the object file (for example, the code was compiled with the **-g** option given to **dmdcc** and the information was not stripped out by a utility or link editor option).

mc68dump

The **mc68dump** utility allows examination of an object file by listing the contents of the file. The dump utility is normally used to look at different parts of an object file, with the parts being selected by options. **mc68dump** attempts to format the information it dumps in a meaningful way by printing certain information in ASCII, hexadecimal, octal, or decimal, as appropriate. The input file is unchanged after execution of **mc68dump**, and no new files are created. **mc68dump** accepts as input both object files and archive libraries of object files.

A simple example of an **mc68dump** is the CCS command

```
mc68dump -t dmda.out
```

that displays the symbol table from the file **dmda.out**. The command

```
mc68dump -tv dmda.out
```

displays the symbol table from the file **dmda.out** in symbolic form. The command

```
mc68dump -f -h -r -t 3 +t 10 test.o >testdump
```

lists the file and section headers, the relocation information, and the symbol table entries three through ten for the object file **test.o**. This command also places the output on the file **testdump**.

mc68ld

The link editor, **mc68ld**, combines object files into one file. To do this, **mc68ld** resolves external symbols. If any argument is a library, it is searched, loading only those routines defining unresolved external references. The library does not have to be ordered using **mc68lorder** because **mc68ld** passes through the library's (archive) symbol table as many times as necessary.

mc68lorder

The CCS command for library ordering, **mc68lorder**, like the other utilities, works the same way as its UNIX System V counterpart.

The output of **mc68lorder** is a list of pairs of object file names, where the first file of the pair contains references to external identifiers defined in the second.

The names of input object files **must** end with **.o**, even when contained in library archives. Files with names not adhering to this rule have their global symbols and references attributed to some other file, and nonsense results.

Other Utilities

The output from **mc68lorder** may be processed by the UNIX System V command **tsort** to find an ordering of a library suitable for one-pass access by the link editor, **mc68ld**. The following example shows the use of **tsort**, along with the archive maintainer, **mc68ar**, to build a new library from all existing files with names ending in **.o**. The archive library is named *libx.a* both before and after the operation.

```
mc68ar cr libx.a 'mc68lorder *.o | tsort'
```

mc68nm

The name list utility, **mc68nm**, displays the symbol table for each MC68000 microprocessor file that is given as input. The input may be a relocatable or an absolute MC68000 microprocessor object file; or it may be an archive library of relocatable or absolute object files.

For each symbol in the table, the following information is printed:

Name: the name of the symbol.

Value: the symbol value expressed as an offset or an address depending on storage class.

Class: the storage class.

Type: the symbol's 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 (for example, "struct-person" where "person" is the structure tag). If the symbol is an array, then the array dimensions are given following the type (for example, char[n][m]).

Size: the size in bytes, if applicable. Special symbols have undefined size.

Line: the source line number where it is defined, if applicable.

Section: for storage classes static and external, the object file section containing the symbol.

mc68size

mc68size prints the number of bytes required for each section (**.text**, **.data**, and **.bss**) of the input MC68000 microprocessor file and the total number of bytes for all sections.

mc68strip

The strip utility removes the symbol table and line number information from MC68000 microprocessor files and archive libraries, thus saving space. The effect of **mc68strip** is the same as the **-s** option of **mc68ld**. After a file has been stripped, no symbolic debugging access is available for that file. This command should be run only on production versions of object files that have been debugged and tested.

Register Use

The MC68000 microprocessor provides sixteen 32-bit, general-purpose registers (d0-d7, a0-a7). The first eight registers (d0-d7) are used as data registers for byte (8-bit), word (16-bit), and long word (32-bit) operations. The next seven registers (a0-a6) and the stack pointer (a7) act as software stack pointers and base address registers. Operations using registers are smaller in size than memory-accessing ones, sometimes by nearly seventy percent.

The high level C language offers a storage class called *register* to take advantage of the corresponding hardware feature. As an example, **dmdcc** generates the following optimized assembler code from two equivalent functions: one function using register variables extensively; the other function using stack variables.

C code:

```
foo1 (a)
register int a;
{
    register int x;
    x = 1;
}
```

```
foo2 (a)
int a;
{
    int x;
    x = 1;
}
```

Assembler code:

#:	assembler:	opcode:
		foo1:
0:	4e56 fff4	link %fp,&-0xc
4:	48ee 000c fff8	movm.l &0xc,-0x8(%fp)
a:	342e 0008	mov.w 0x8(%fp),%d2
e:	7601	movq.l &0x1,%d3

```

10: 4cee 000c fff8    movm.l -0x8(%fp),&0xc
16: 4e5e                unlk  %fp
18: 4e75                rts

                                foo2:
0:  4e56 fffa          link  %fp,&-0x6
4:  3d7c 0001 fffe     mov.w  &0x1,-0x2(%fp)
a:  4e5e                unlk  %fp
c:  4e75                rts

```

It can be seen that:

1. There is an overhead by using register variables in the initialization step. Two *movm.l* instructions are needed to implement the transparency principle of saving register contents on entry and restoring them on exit. In addition, if any function argument is declared as *register*, there is a corresponding *mov.w* instruction to store that value into a physical register.
2. There is substantial code reduction in the body of the function. In the example above, "*movq.l &0x1,%d3*" assembles to 2 bytes while the equivalent non-register instruction "*mov.w &0x1,-0x2(%fp)*" uses 6 bytes, so a size reduction of almost 70 percent is achieved.
3. There is a limit to the number of declared *register* variables that are actually stored into hardware registers. From the example above, *d2* is used as the first data register; therefore, the user can use five more registers (*d3-d7*) to store integer variables. Additional data *register* declarations will be ignored, and the variables will be stored into memory (stack). The following table lists all the MC68000 general-purpose registers and their usage availability.

Register Use

Registers	Number	Usage
d0-d1	2	Scratch data registers (N/A)
d2-d7	6	Data register variables
a0-a1	2	Scratch address registers (N/A)
a2-a5	4	Register pointer variables
a6-a7	2	Frame and stack pointers (N/A)

The previous observations suggest the following conclusions concerning register variable usage for code reduction:

1. Do not use register variables when the initialization overhead is larger than the gain in the function body. This applies to:
 - any function argument that is used only once in the function body (you save the "mov.w" instruction)
 - an automatic variable inside a function that has only one such variable and uses it only once (you save the two "mov.m" instructions).
2. Declare as *register* any heavily used variable. Once this is done, the code for register transparency is generated, and declaring additional variables as *register* can only improve code size. The rule is: when *register* has been used once, use it all the time from then on.
3. Do not restrict register usage to simple variables. Global and complex variables that are heavily referenced may be conveniently stored into declared *register* variables for optimal access. Note that this method works only if the variable being substituted is referred (i.e., its value is used [read] but not updated [write]) throughout the function. For example:

```
struct date { int day; int month; int year};

int newyear;

setdate (d)
register struct date *d;
{ register int myday = d->day;      /* substitute for complex variable */
  register int myyear = newyear;   /* substitute for global variable */
  register int mymonth = d->month; /* potential error here           */

  ...          /* "myday" and "myyear" must be referred several times */
              /* in order to recover the overhead of initializing them */

  d->month++; /* now, "mymonth" and "d->month" have
              different values !!! */
}
```

4. Identify function arguments and automatic variables as data (*long*, *int*, *short*) variables or pointer variables. In each category, rank them according to usage criteria. The most heavily-used variables should be declared first, so they can be physically assigned to a hardware register.

C Language

The 630 MTG CCS compiles the C language described in the Kernighan and Ritchie book* with the following enhancements:

1. The **dmdcc** compiler includes recent enhancements to the C language. These enhancements allow structures to be assigned or copied as a unit and to be passed to and returned from functions.
2. The compiler does not have the restriction that only eight characters of a variable name are significant. The entire name is significant.

On the MC68000 microprocessor, C language data types map in the natural way for a 16-bit processor; that is, **char** maps to the type byte, **int** and **short** map to 16-bit word, and **long** maps to 32-bit double word. The **dmdcc** compiler implements both the **float** and **double** data type specifiers using 32-bit precision floating point arithmetic.

Standard C language leaves identification of the assembler escape keyword *asm* to the system designer. *asm* has been implemented for **dmdcc** with the syntax *asm* ("assembly instruction "). For example, *asm* ("mov.1 &0,%d0 ") loads data register zero with a zero. The assembly language instruction contained within the quotation marks is transmitted unchanged to the assembler.

* B. W. Kernighan and D. M. Ritchie, *The C Programming Language* (Englewood Cliffs, New Jersey: Prentice-Hall, 1978)

Chapter 16: MC68000 UNIX System Assembler

Introduction	16-1
Warnings	16-2
Comparison Instructions	16-2
Overloading of Opcodes	16-3
Use of the Assembler	16-4
General Syntax Rules	16-5
Format of Assembly Language Line	16-5
Comments	16-6
Identifiers	16-6
Register Identifiers	16-7
Constants	16-7
Numerical Constants	16-7
Character Constants	16-8
Other Syntactic Details	16-8
Segments, Location Counters, and Labels	16-9
Segments	16-9
Location Counters and Labels	16-10
Types	16-11
Expressions	16-12
Pseudo-Operations	16-13
Data Initialization Operations	16-13
Symbol Definition Operations	16-14
Location Counter Control Operations	16-15

Chapter 16: MC68000 UNIX System Assembler

Symbolic Debugging Operations 16-16
 “file” and “ln” 16-16
 Symbol Attribute Operations 16-16
 Switch Table Operation 16-18

Span-Dependent Optimization 16-19

Address Mode Syntax 16-21

Machine Instructions 16-24

Introduction

This chapter is a reference manual for **MC68AS**, the assembler language interpreted by the **mc68as** assembler. The **mc68as** assembler is a component of the 630 MTG C Compilation System.

Programmers familiar with the MC68000 processor should be able to program in **MC68AS** by referring to this chapter, but this is not a manual for the processor itself. Details about the effects of instructions, meanings of status register bits, handling of interrupts, and many other issues are not dealt with here. This chapter, therefore, should be used in conjunction with a MC68000 processor handbook such as the "*MC68000 16-Bit Microprocessor User's Manual*".

* "*MC68000 16-Bit Microprocessor User's Manual*," Third Edition; Englewood Cliffs, N.J.: Prentice-Hall, 1982

Warnings

A few important warnings to the **MC68AS** user should be emphasized at the outset. Though for the most part there is a direct correspondence between **MC68AS** notation and the notation used in the *MC68000 User's Manual*, the following exceptions could lead the unsuspecting user to write an incorrect code.

Comparison Instructions

First, the order of the operands in *compare* instructions follows one convention in the *MC68000 16-Bit Microprocessor User's Manual* and the opposite convention in **MC68AS**. Using the convention of the *User's Manual*, you might write:

```
CMP.W   D5,D3           Is D3 less than D5 ?  
BLE     IS_LESS        Branch if less.
```

Using the **MC68AS** convention, you would write:

```
cmp.w   %d3,%d5        # Is d3 less than d5 ?  
ble     is_less        # Branch if less.
```

MC68AS follows the convention used by other assemblers supported in the UNIX System. This convention makes for straightforward reading of compare-and-branch instruction sequences, but does nonetheless lead to the peculiarity that if a *compare* instruction is replaced by a *subtract* instruction, the effect on the condition codes will be entirely different. This may be confusing to programmers who are used to thinking of a comparison as a subtraction whose result is not stored. But users of **MC68AS** who become accustomed to the convention will find that both the *compare* and *subtract* notations make sense in their respective contexts.

Overloading of Opcodes

Another issue that users must be aware of arises from the MC68000's use of several different instructions to do more or less the same thing. For example, the MC68000 User's Manual lists the instructions **SUB**, **SUBA**, **SUBI**, and **SUBQ**, which all have the effect of subtracting their source operand from their destination operand. **MC68AS** provides the convenience of allowing all these operations to be specified by a single assembly instruction **sub**. On the basis of the operands given to the **sub** instruction, the **MC68AS** assembler selects the appropriate MC68000 operation code.

The danger created by this convenience is that it could leave the misleading impression that all forms of the **SUB** operation are semantically identical when, in fact, they are not. The careful reader of the MC68000 User's Manual will notice that whereas **SUB**, **SUBI**, and **SUBQ** all affect the condition codes in a consistent way, **SUBA** does not affect the condition codes at all. Consequently, the **MC68AS** user must be aware that when the destination of a **sub** instruction is an address register (which causes the **sub** to be mapped into the operation code for **SUBA**), the condition codes will not be affected.

Use of the Assembler

The UNIX System command **mc68as** invokes the assembler and has the following syntax:

```
mc68as [ -o output ] file
```

This causes the named file to be assembled. The output of the assembly is left in the file *output* specified with the *-o* flag. If no such specification is made, the output is left in the file whose name is formed by removing the *.s* suffix, if there is one, from the input file name and appending a *.o* suffix. For more information on **mc68as**, see the manual page **MC68AS(1)** in the *630 MTG Software Reference Manual*.

General Syntax Rules

Format of Assembly Language Line

Typical lines of MC68AS assembly code look like these:

```
# Clear a block of memory at location %a3

        text      2
        mov.w     &const,%d1
loop:   clr.l     (%a3)+
        dbf      %d1,loop      # go back for const
                                   # repetitions

init2:
        clr.l    count; clr.l credit; clr.l debit;
```

These general points about the example should be noted:

- An identifier occurring at the beginning of a line and followed by a colon (:) is a *label*. One or more labels may precede any assembly language instruction or pseudo-operation. See also "Location Counters and Labels" in this chapter.
- A line of assembly code need not include an instruction. It may consist of a comment alone (introduced by #), a label alone (terminated by :), or it may be entirely blank.
- It is good practice to use tabs to align assembly language operations and their operands into columns, but this is *not* a requirement of the assembler. An opcode may appear at the beginning of the line, if desired, and spaces may precede a label. A single blank or tab suffices to separate an opcode from its operands. Additional blanks and tabs are ignored by the assembler.
- It is permissible to write *several* instructions on one line by separating them by semicolons. The semicolon is syntactically equivalent to a new line. But a semicolon inside a comment is ignored.

Comments

Comments are introduced by the character # and continue to the end of the line. Comments may appear anywhere and are completely disregarded by the assembler.

Identifiers

An identifier is a string of characters taken from the set *a-z, A-Z, _, <, and 0-9*. The first character of an identifier must be a letter (upper or lower case) or an underscore. Upper and lower case letters are distinguished;

con35 and *CON35*

are two distinct identifiers.

There is no limit on the length of an identifier.

The value of an identifier is established by the **set** pseudo-operation (see "Symbol Definition Operations" in this chapter) or by using it as a label (see "Location Counters and Labels" in this chapter).

The character < has special significance to the assembler. A < used alone, as an identifier, means "the current location." A < used as the first character in an identifier becomes a "." in the symbol table, allowing symbols such as *.eos* and *.Ofake* to make it into the symbol table, as required by the Common Object File Format.*

* See the UNIX System V Programmer Reference Manual

Register Identifiers

A register identifier is an identifier preceded by % the character and represents one of the MC68000 microprocessor's registers. The predefined register identifiers are:

<code>%d0</code>	<code>%d4</code>	<code>%a0</code>	<code>%a4</code>	<code>%cc</code>	<code>%usp</code>
<code>%d1</code>	<code>%d5</code>	<code>%a1</code>	<code>%a5</code>	<code>%pc</code>	<code>%fp</code>
<code>%d2</code>	<code>%d6</code>	<code>%a2</code>	<code>%a6</code>	<code>%sp</code>	
<code>%d3</code>	<code>%d7</code>	<code>%a3</code>	<code>%a7</code>	<code>%sr</code>	

Important Note: The identifiers `%a7` and `%sp` represent one and the same machine register. Likewise, `%a6` and `%fp` are equivalent. Use of both `%a7` and `%sp`, or `%a6` and `%fp`, in the same program may result in confusion.

Constants

MC68AS deals only with integer constants. They may be entered in decimal, octal, or hexadecimal, or they may be entered as character constants. Internally, MC68AS treats all constants as 32-bit binary two's complement quantities.

Numerical Constants

A decimal constant is a string of digits beginning with a non-zero digit.

An octal constant is a string of digits beginning with zero.

A hexadecimal constant consists of the characters `0x` or `0X` followed by a string of characters from the set `0-9`, `a-f`, and `A-F`. In hexadecimal constants upper and lower case letters are not distinguished.

Examples:

<code>set</code>	<code>const,35</code>	<code># Decimal 35</code>
<code>mov.w</code>	<code>&035,%d1</code>	<code># Octal 35 (decimal 29)</code>
<code>set</code>	<code>const,0x35</code>	<code># Hex 35 (decimal 53)</code>
<code>mov.w</code>	<code>&0xff,%d1</code>	<code># Hex ff (decimal 255)</code>

Character Constants

An ordinary character constant consists of a single-quote (') followed by an arbitrary ASCII character other than \. The value of the constant is equal to the ASCII code for the character. Special meanings of characters are overridden when used in character constants; for example, if # is used, the # is not treated as introducing a comment.

A special character constant consists of ' \ followed by another character. All the special character constants and examples of ordinary character constants are listed here:

<i>Constant</i>	<i>Value</i>	<i>Meaning</i>
'\b	0x08	Backspace
'\t	0x09	Horizontal Tab
'\n	0x0a	Newline (Line Feed)
'\v	0x0b	Vertical Tab
'\f	0x0c	Form Feed
'\r	0x0d	Carriage Return
'\\	0x5c	Backslash (\)
'	0x27	Single-Quote
'0	0x30	Zero
'A	0x41	Capital A
'a	0x61	Lower Case A

Other Syntactic Details

A discussion of expression syntax appears under "Expressions" in this chapter. Information about the syntax of specific components of MC68AS instructions and pseudo-operations is given under "Pseudo-Operations," "Address Mode Syntax," and "Machine Instructions" in this chapter.

Segments, Location Counters, and Labels

Segments

A program in MC68AS assembly language may be broken into segments known as *text*, *data*, and *bss* segments. The convention regarding the use of these segments is to place instructions in *text* segments, initialized data in *data* segments, and uninitialized data in *bss* segments. However, the assembler does not enforce this convention; for example, it permits intermixing of instructions and data in a *text* segment.

Primarily to simplify compiler code generation, the assembler permits up to four separate *text* segments and four separate *data* segments named 0, 1, 2, and 3. The assembly language program may switch freely between them by using assembler pseudo-operations (see "Location Counter Control Operations" in this chapter). When generating the object file, the assembler concatenates the *text* segments to generate a single *text* segment, and the *data* segments to generate a single *data* segment. Thus, the object file contains only one *text* segment and only one *data* segment.

There is only one *bss* segment to begin with, and it maps directly into the object file.

Because the assembler keeps together everything from a given segment when generating the object file, the order in which information appears in the object file may not be the same as in the assembly language file. For example, if the data for a program consisted of:

```
data    1                # segment 1
word    0x1111
data    0                # segment 0
long    0xffffffff
data    1                # segment 1
byte    0x2222
```

Segments, Location Counters, and Labels

then equivalent object code would be generated by:

```
data    0
long    0xffffffff
word    0x1111
word    0x2222
```

Location Counters and Labels

The assembler maintains separate *location counters* for the *bss* segment and for each of the *text* and *data* segments. The location counter for a given segment is incremented by one for each byte generated in that segment.

The location counters allow values to be assigned to labels. When an identifier is used as a label in the assembly language input, the current value of the current location counter is assigned to the identifier. The assembler also keeps track of which segment the label appeared in. Thus, the identifier represents a memory location relative to the beginning of a particular segment.

Types

Identifiers and expressions may have values of different types.

- In the simplest case, an expression (or identifier) may have an *absolute* value, such as 29, -5000, or 262143.
- An expression (or identifier) may have a value relative to the start of a particular segment. Such a value is known as a *relocatable* value. The memory location represented by such an expression cannot be known at assembly time, but the relative values (i.e., the difference) of two such expressions can be known if they refer to the same segment.

Identifiers which appear as labels have *relocatable* values.

- If an identifier is never assigned a value, it is assumed to be an *undefined external*. Such identifiers may be used with the expectation that their values will be defined in another program, and hence known at load time; but the relative values of *undefined externals* cannot be known.

Expressions

For conciseness, the following abbreviations will be useful:

abs *absolute expression*
rel *relocatable expression*
ext *undefined external*

All constants are absolute expressions. An identifier may be thought of as an expression having the identifier's type. Expressions may be built up from lesser expressions using the operators +, -, *, and /, according to the following type rules:

$abs + abs = abs$
 $abs + rel = rel + abs = rel$
 $abs + ext = ext + abs = ext$

$abs - abs = abs$
 $rel - abs = rel$
 $ext - abs = ext$
 $rel - rel = abs,$

*provided that the two relocatable expressions
are relative to the same segment*

$abs * abs = abs$

$abs / abs = abs$

$- abs = abs$

Important Note: Use of a *rel-rel* expression is dangerous, particularly when dealing with identifiers from *text* segments. The problem is that the assembler will determine the value of the expression before it has resolved all questions concerning span-dependent optimizations. Use this feature at your own risk!

The unary minus operator takes the highest precedence; the next highest precedence is given to * and /, and the lowest precedence is given to + and binary -. Parentheses may be used to coerce the order of evaluation.

If the result of a division is a positive non-integer, it will be truncated toward zero. If the result is a negative non-integer, the direction of truncation cannot be guaranteed.

Pseudo-Operations

Data Initialization Operations

byte *abs, abs, ...*

One or more arguments, separated by commas, may be given. The values of the arguments are computed to produce successive bytes in the assembly output.

short *abs, abs, ...*

One or more arguments, separated by commas, may be given. The values of the arguments are computed to produce successive 16-bit words in the assembly output.

long *expr, expr, ...*

One or more arguments, separated by commas, may be given. Each expression may be *absolute*, *relocatable*, or *undefined external*. A 32-bit quantity is generated for each such argument (in the case of *relocatable* or *undefined external* expressions, the actual value may not be filled in until load time).

Alternatively, the arguments may be bit-field expressions. A bit-field expression has the form

$$n : value$$

where both *n* and *value* denote *absolute* expressions. The quantity *n* represents a field width; the low-order *n* bits of *value* become the contents of the bit-field. Successive bit-fields fill up 32-bit long quantities starting with the high-order part. If the sum of the lengths of the bit-fields is less than 32 bits, the assembler creates a 32-bit long with zeroes filling out the low-order bits. For example,

Pseudo-Operations

long 4:-1, 16:0x7f, 12:0, 5000

and

long 4:-1, 16:0x7f, 5000

are equivalent to

long 0xf007f000, 5000

Bit-fields may not span pairs of 32-bit longs. Thus,

long 24:0xa, 24:0xb, 24:0xc

yields the same thing as

long 0x00000a00, 0x00000b00, 0x00000c00

space *abs* The value of *abs* is computed, and the resultant number of bytes of zero data is generated. For example,

space 6

is equivalent to

byte 0,0,0,0,0,0

Symbol Definition Operations

set *identifier, expr*

The value of *identifier* is set equal to *expr*, which may be absolute or relocatable.

comm *identifier, abs*

The named *identifier* is to be assigned to a common area of size *abs* bytes. If *identifier* is not defined by another program, the loader will allocate space for it.

The type of *identifier* becomes *undefined external*.

lcomm *identifier, abs*

The named identifier is assigned to a *local common* of size *abs* bytes. This results in allocation of space in the *bss* segment.

The type of *identifier* becomes *relocatable*.

global *identifier*

This causes *identifier* to be externally visible. If *identifier* is defined in the current program, then declaring it global allows the loader to resolve references to *identifier* in other programs.

If *identifier* is not defined in the current program, the assembler expects an external resolution; in this case, *identifier* is global by default.

Location Counter Control Operations

data *abs* The argument, if present, must evaluate to 0, 1, 2, or 3; this indicates the number of the data segment into which assembly is to be directed. If no argument is present, assembly is directed into data segment 0.

text *abs* The argument, if present, must evaluate to 0, 1, 2, or 3; this indicates the number of the text segment into which assembly is to be directed. If no argument is present, assembly is directed into text segment 0.

Before the first **data** or **text** operation is encountered, assembly is by default directed into text segment 0.

org *expr* The current location counter is set to *expr*. *Expr* must represent a value in the current segment and must not be less than the current location counter.

even The current location counter is rounded up to the next even value.

Symbolic Debugging Operations

The assembler allows for symbolic debugging information to be placed into the object code file with special pseudo-operations. The information typically includes line numbers and information about C language symbols, such as their type and storage class. The 630 MTG CCS C compiler generates symbolic debugging information when the **-g** option is used. Assembler programmers may also include such information in source files.

“file” and “ln”

The **file** pseudo-operation passes the name of the source file into the object file symbol table. It has the form

```
file "filename"
```

where *filename* consists of one to 14 characters.

The *ln* pseudo-operation makes a line number table entry in the object file. That is, it associates a line number with a memory location. Usually the memory location is the current location in text. The format is

```
ln line [,value]
```

where **line** is the line number. The optional **value** is the address in text, data, or bss to associate with the line number. The default when **value** is omitted (which is usually the case) is the current location in text.

Symbol Attribute Operations

The basic symbolic testing pseudo-operations are **def** and **endef**. These operations enclose other pseudo-operations that assign attributes to a symbol and must be paired.

```
def    name
.      # Attribute
.      # Assigning
.      # Operations
endef
```

Note 1: **def** does not define the symbol, although it does create a symbol table entry. Because an undefined symbol is treated as external, a symbol which appears in a **def**, but which

never acquires a value, will ultimately result in an error at link edit time.

Note 2: To allow the assembler to calculate the sizes of functions for other CCS tools, each **def/edef** pair that defines a function name must be matched by a **def/edef** pair after the function in which a storage class of **-1** is assigned.

The paragraphs below describe the attribute-assigning operations. Keep in mind that all of these operations apply to symbol *name* which appeared in the opening **def** pseudo-operation.

- val** *expr* Assigns the value *expr* to *name*. The type of the expression *expr* determines with which section *name* is associated. If value is , the current location in the text section is used.
- scl** *expr* Declares a storage class for *name*. The expression *expr* must yield an ABSOLUTE value that corresponds to the C compiler's internal representation of a storage class. The special value **-1** designates the physical end of a function.
- type** *expr* Declares the C language type of *name*. The expression *expr* must yield an ABSOLUTE value that corresponds to the C compiler's internal representation of a basic or derived type.
- tag** *str* Associates *name* with the structure, enumeration, or union named *str* which must have already been declared with a **def/edef** pair.
- line** *expr* Provides the line number of *name*, where *name* is a block symbol. The expression *expr* should yield an ABSOLUTE value that represents a line number.
- size** *expr* Gives a size for *name*. The expression *expr* must yield an ABSOLUTE value. When *name* is a structure or an array with a predetermined extent, *expr* gives the size in bytes. For bit fields, the size is in bits.
- dim** *expr1, expr2, ...*
Indicates that *name* is an array. Each of the expressions must

yield an ABSOLUTE value that provides the corresponding array dimension.

Switch Table Operation

The 630 MTG CCS C compiler generates a compact set of instructions for the C language *switch* construct, of which an example is shown below.

```
sub.l  &1, %d0
cmp.l  %d0, &4
bhi    L%21
add.w  %d0, %d0
mov.w  10(%pc, %d0.w), %d0
jmp    6(%pc, %d0.w)
swbeg  &5
L%22:
short  L%15-L%22
short  L%21-L%22
short  L%16-L%22
short  L%21-L%22
short  L%17-L%22
```

The special **swbeg** pseudo-operation communicates to the assembler that the lines following it contain *rel-rel* subtractions. Remember that ordinarily such subtractions are risky because of span-dependent optimization. In this case, however, the assembler makes special allowances for the subtraction because the compiler guarantees that both symbols will be defined in the current assembler file, and that one of the symbols is a fixed distance away from the current location.

The **swbeg** pseudo-operation takes an argument that looks like an immediate operand. The argument is the number of lines that follow **swbeg** and that contain switch table entries. **Swbeg** inserts two words into text. The first is the ILLEGAL instruction code. The second is the number of table entries that follow. The 630 MTG CCS disassembler needs the ILLEGAL instruction as a hint that what follows is a switch table. Otherwise, the disassembler would get confused when trying to decode the table entries (differences between two symbols) as instructions.

Span-Dependent Optimization

The assembler makes certain choices about the object code it generates based on the distance between an instruction and its operand(s). Choosing the smallest, fastest form is called span-dependent optimization. Span-dependent optimization occurs most obviously in the choice of object code for branches and jumps. It also occurs when an operand may be represented by the program counter relative address mode instead of as an absolute 2-word (long) address. The span-dependent optimization capability is normally enabled; the `-n` command line flag disables it. When this capability is disabled, the assembler makes worst-case assumptions about the types of object code that must be generated.

In the 630 MTG CCS, the compiler generates branch instructions without a specific offset size. When the optimizer is used, it identifies branches which could be represented by the short form, and it changes the operation accordingly. The assembler chooses only between long and very long representations for branches.

Branch instructions, e.g., `bra`, `bsr`, `bgt`, etc., can have either a byte or a word pc-relative address operand. A byte-size specification should be used only when the user is sure that the address intended can be represented in the byte allowed. The assembler will take one of these instructions with a byte-size specification and generate the byte form of the instruction without asking questions.

Although the largest offset specification allowed is a word, large programs could conceivably have need for a branch to a location not reachable by a word displacement. Therefore, equivalent long forms of these instructions might be needed. When the assembler encounters a branch instruction without a size specification, or with a word size specification, it tries to choose between the long and very long forms of the instruction. If the operand can be represented in a word, then the word form of the instruction will be generated. Otherwise, the very long form will be generated. For unconditional branches, e.g., `br`, `bra` and `bsr`, the very long form is just the equivalent jump (`jmp` and `jsr`) with an absolute address operand (instead of pc-relative). For conditional branches, the equivalent very long form is a conditional branch around a jump, where the conditional test has been reversed.

Span-Dependent Optimization

Figure 16-1 summarizes span-dependent optimizations. The assembler chooses only between the long form and very long form, while the optimizer chooses between the short and long forms for branches (but not bsr).

Instruction	Short Form	Long Form	Very Long Form
br, bra, bsr	byte offset	word offset	jmp or jsr with absolute long address
conditional branch	byte offset	word offset	short conditional branch with reversed condition around jmp with absolute long address
jmp, jsr	—	pc-relative address	absolute long address
lea.l, pea.l	—	pc-relative address	absolute long address

Figure 16-1: Assembler Span-Dependent Optimizations

Address Mode Syntax

Figure 16-2 summarizes the **MC68AS** syntax for MC68000 addressing modes.

In the table, the letter *n* represents any digit from 0 to 7. The notations *R_i* and *ri* represent any of the MC68000 data or address registers.

The letter *d*, when used to represent a displacement, may stand for any absolute expression.

It is important to note that expressions used for the **Absolute** addressing modes need not be *absolute expressions* in the sense defined under "Types" in this chapter. Although the addresses used in those addressing modes must ultimately be filled in with constants (which can be done by the loader) there is no need for the assembler to be able to compute them. Indeed, the **Absolute Long** addressing mode is commonly used for accessing *undefined external* addresses.

Address Mode Syntax

Motorola Notation	MC68AS Notation	Effective Address Mode
Dn	%dn	Data Register Direct
An	%an	Address Register Direct
(An)	(%an)	Address Register Indirect
An@+	(%an)+	Address Register Indirect with Postincrement
An@-	-(%an)	Address Register Indirect with Predecrement
An@(d)	d(%an)	Address Register Indirect with Displacement (<i>d</i> signifies a signed 16-bit absolute displacement)
An@(d,Ri.W) An@(d,Ri.L)	d(%an,%ri.w) d(%an,%ri.l)	Address Register Indirect with Index (<i>d</i> signifies a signed 8-bit absolute displacement)
xxx.W	xxx	Absolute Short Address (<i>xxx</i> signifies an expression yielding a signed 16-bit memory address)
xxx.L	xxx	Absolute Long Address (<i>xxx</i> signifies an expression yielding a 32-bit memory address)

Figure 16-2: Effective Address Modes (Sheet 1 of 2)

Motorola Notation	MC68AS Notation	Effective Address Mode
PC@(d)	d(%pc)	Program Counter with Displacement (<i>d</i> signifies a signed 16-bit absolute displacement)
PC@(d,Ri.W) PC@(d,Ri.L)	d(%pc,%rn.w) d(%pc,%rn.l)	Program Counter with Index (<i>d</i> signifies a signed 8-bit absolute displacement)
#xxx	&xxx	Immediate Data (<i>xxx</i> signifies an absolute constant expression)

Figure 16-2: Effective Address Modes (Sheet 2 of 2)

Machine Instructions

Figure 16-3 shows how MC68000 instructions should be written in order to be understood correctly by the **MC68AS** assembler. Several abbreviations are used in the table:

- S The letter *S*, as in *add.S*, stands for one of the operation size attribute letters *b*, *w*, or *l*, representing a byte, word, or long operation.
- A The letter *A*, as in *add.A*, stands for one of the address operation size attribute letters *w* or *l*, representing a word or long operation.
- CC In the contexts *bCC*, *dbCC*, and *sCC*, the letters *CC* represent any of the following condition code designations (except that *f* and *t* may not be used in the *bCC* instruction):

cc	carry clear	ls	low or same
cs	carry set	lt	less than
eq	equal	mi	minus
f	false	ne	not equal
ge	greater or equal	pl	plus
gt	greater than	t	true
hi	high	vc	overflow clear
hs	high or same (=cc)	vs	overflow set
le	less or equal		
lo	low (=cs)		

- D An absolute expression evaluating to a 16-bit displacement.
 - EA This represents an arbitrary effective address.
 - I An absolute expression, used as an immediate operand.
 - Q An absolute expression evaluating to a number from 1 to 8.
 - L A label reference, or any expression representing a memory address in the current segment.
- %dx*, *%dy*, *%dn*, *%ax*, *%ay*, and *%an* These represent registers.

Operation	MC68AS Syntax	Meaning
ABCD	abcd.b %dy,%dx -(%ay),-(%ax)	Add Decimal with Extend
ADD	add.S EA,%dn %dn,EA	Add Binary
ADDA	add.A EA,%an	Add Address
ADDI	add.S &I,EA	Add Immediate
ADDQ	add.S &Q,EA	Add Quick
ADDX	addx.S %dy,%dx -(%ay),-(%ax)	Add Extended
AND	and.S EA,%dn %dn,EA	AND Logical
ANDI	and.S &I,EA	AND Immediate
ANDI to CCR	and.b &I,%cc	AND Immediate to Condition Codes
ANDI to SR	and.w &I,%sr	AND Immediate to the Status Register

Figure 16-3: MC68000 Instruction Formats (Sheet 1 of 9)

Machine Instructions

Operation	MC68AS Syntax	Meaning
ASL	asl.S %dx,%dy &Q,%dy	Arithmetic Shift (Left)
	asl.w &1,EA	
ASR	asr.S %dx,%dy &Q,%dy	Arithmetic Shift (Right)
	asr.w &1,EA	
Bcc	bCC L	Branch Conditionally (16-bit Displacement)
	bCC.b L	Branch Conditionally (Short) (8-bit Displacement)
BCHG	bchg %dn,EA &I,EA	Test a Bit and Change Note: bchg should be written with no suffix. If the second operand is a data register, .l is assumed; otherwise .b is.
BCLR	bclr %dn,EA &I,EA	Test a Bit and Clear Note: bclr should be written with no suffix. If the second operand is a data register, .l is assumed; otherwise .b is.

Figure 16-3: MC68000 Instruction Formats (Sheet 2 of 9)

Operation	MC68AS Syntax	Meaning
BRA	bra L	Branch Always (16-bit Displacement)
	bra.b L	Branch Always (Short) (8-bit Displacement)
	br L br.b L	Same as bra Same as bra.b
BSET	bset %dn,EA &I,EA	Test a Bit and Set Note: bset should be written with no suffix. If the second operand is a data register, .l is assumed; otherwise .b is.
BSR	bsr L	Branch to Subroutine (16-bit Displacement)
	bsr.b L	Branch to Subroutine (Short) (8-bit Displacement)
BTST	btst %dn,EA &I,EA	Test a Bit Note: btst should be written with no suffix. If the second operand is a data register, .l is assumed; otherwise .b is.
CHK	chk.w EA,%dn	Check Register Against Bounds
CLR	clr.S EA	Clear an Operand

Figure 16-3: MC68000 Instruction Formats (Sheet 3 of 9)

Machine Instructions

Operation	MC68AS Syntax	Meaning
CMP	cmp.S %dn,EA	Compare
CMPA	cmp.A %an,EA	Compare Address
CMPI	cmp.S EA,&I	Compare Immediate
CMPM	cmp.S (%ax)+,(%ay)+	Compare Memory
		Note: The order of operands in MC68AS is the reverse of that in the MC68000 User's Manual.
DBcc	dbCC %dn,L	Test Condition, Decrement, and Branch
	dbra %dn,L	Decrement and Branch Always
	dbr %dn,L	Same as dbra
DIVS	divs.w EA,%dn	Signed Divide
DIVU	divu.w EA,%dn	Unsigned Divide
EOR	eor.S %dn,EA	Exclusive OR Logical
EORI	eor.S &I,EA	Exclusive OR Immediate
EORI to CCR	eor.b &I,%cc	Exclusive OR Immediate to Condition Codes
EORI to SR	eor.w &I,%sr	Exclusive OR Immediate to the Status Register
EXG	exg %rx,%ry	Exchange Registers

Figure 16-3: MC68000 Instruction Formats (Sheet 4 of 9)

Operation	MC68AS Syntax	Meaning
EXT	ext.A %dn	Sign Extend
JMP	jmp EA	Jump
JSR	jsr EA	Jump to Subroutine
LEA	lea.l EA,%an	Load Effective Address
LINK	link %an,&I	Link and Allocate
LSL	lsl.S %dx,%dy &Q,%dy	Logical Shift (Left)
	lsl.w &1,EA	
LSR	lsr.S %dx,%dy &Q,%dy	
	lsr.w &1,EA	

Figure 16-3: MC68000 Instruction Formats (Sheet 5 of 9)

Machine Instructions

Operation	MC68AS Syntax	Meaning
MOVE	mov.S EA,EA	Move Data from Source to Destination Note: If the destination is an address register, the instruction generated is MOVEA.
MOVE to CCR	mov.w EA,%cc	Move to Condition Codes
MOVE to SR	mov.w EA,%sr	Move to the Status Register
MOVE from SR	mov.w %sr,EA	Move from the Status Register
MOVE USP	mov.l %usp,%an %an,%usp	Move User Stack Pointer
MOVEA	mov.A EA,%an	Move Address
MOVEM	movm.A &I,EA EA,&I	Move Multiple Registers Note: The immediate operand is a mask designating which registers are to be moved to memory or which registers are to receive memory data. Not all addressing modes are permitted, and the correspondence between mask bits and register numbers depends on the addressing mode used! See MC68000 User's Manual for details.

Figure 16-3: MC68000 Instruction Formats (Sheet 6 of 9)

Operation	MC68AS Syntax	Meaning
MOVEP	movp.A %dx,D(%ay) D(%ax),%dy	Move Peripheral Data
MOVEQ	mov.l &I,%dn	Move Quick (when I fits in one byte)
MULS	muls.w EA,%dn	Signed Multiply
MULU	mulu.w EA,%dn	Unsigned Multiply
NBCD	nbcd.b EA	Negate Decimal with Extend
NEG	neg.S EA	Negate
NEGX	negx.S EA	Negate with Extend
NOP	nop	No Operation
NOT	not.S EA	Logical Complement
OR	or.S EA,%dn %dn,EA	Inclusive OR Logical
ORI	or.S &I,EA	Inclusive OR Immediate
ORI to CCR	or.b &I,%cc	Inclusive OR Immediate to Condition Codes
ORI to SR	or.w &I,%sr	Inclusive OR Immediate to the Status Register

Figure 16-3: MC68000 Instruction Formats (Sheet 7 of 9)

Machine Instructions

Operation	MC68AS Syntax	Meaning
PEA	pea.l EA	Push Effective Address
RESET	reset	Reset External Devices
ROL	rol.S %dx,%dy &Q,%dy	Rotate (without Extend) (Left)
	rol.w &1,EA	
ROR	ror.S %dx,%dy &Q,%dy	Rotate (without Extend) (Right)
	ror.w &1,EA	
ROXL	roxl.S %dx,%dy &Q,%dy	Rotate with Extend (Left)
	roxl.w &1,EA	
ROXR	roxr.S %dx,%dy &Q,%dy	Rotate with Extend (Right)
	roxr.w &1,EA	
RTE	rte	Return from Exception
RTR	rtr	Return and Restore Condition Codes
RTS	rts	Return from Subroutine

Figure 16-3: MC68000 Instruction Formats (Sheet 8 of 9)

Operation	MC68AS Syntax	Meaning
SBCD	sbcd.b %dy,%dx -(%ay),-(%ax)	Subtract Decimal with Extend
Scc	sCC.b EA	Set According to Condition
STOP	stop &I	Load Status Register and Stop
SUB	sub.S EA,%dn %dn,EA	Subtract Binary
SUBA	sub.A EA,%an	Subtract Address
SUBI	sub.S &I,EA	Subtract Immediate
SUBQ	sub.S &Q,EA	Subtract Quick
SUBX	subx.S %dy,%dx -(%ay),-(%ax)	Subtract with Extend
SWAP	swap.w %dn	Swap Register Halves
TAS	tas.b EA	Test and Set an Operand
TRAP	trap &I	Trap
TRAPV	trapv	Trap on Overflow
TST	tst.S EA	Test an Operand
UNLK	unlk %an	Unlink

Figure 16-3: MC68000 Instruction Formats (Sheet 9 of 9)

Appendix A: Installation and Administration

General Considerations	A-1
Dependencies	A-1
Windowing Utilities	A-1
C++ Programming Language	A-1
Storage Requirements	A-2
Location	A-2
Time-out Feature	A-2
Processes	A-3
Permission Modes	A-3
Installation of 630 MTG Software Development Package	A-4
Installation Procedure for 3B2 Computers	A-4
Installation Procedure for 3B15, 3B4000, and 3B20 Computers	A-5
Windowing Utilities Installation	A-5
UNIX Systems Distributed with AT&T Windowing Utilities	A-5
UNIX Systems Distributed without AT&T Windowing Utilities	A-5
Building Source Code	A-8
Local Software	A-10

General Considerations

This appendix contains general information and instructions for loading, installing, and maintaining the 630 MTG Software Development Package for the 630 MTG. The package should be installed by your UNIX System Administrator.

Dependencies

Windowing Utilities

The 630 MTG is most powerful when used with the AT&T Windowing Utilities Package. This package provides the `xt` UNIX System device driver, the `layers` window manager, and support programs necessary for operating the terminal in the `layers` environment. The `layers` environment allows seven terminal sessions over a single physical host connection.

The AT&T Windowing Utilities Package is supplied with UNIX System V Release 3 on AT&T computers (Release 2.1.1 and beyond for the AT&T 3B20 Computers).

If your computer does not have the AT&T Windowing Utilities Package, you can use the 5620 DMD V2.0-Core Package to provide the `layers` environment for the 630 MTG. See "Windowing Utilities Installation" in this chapter for detailed instructions.

C++ Programming Language

The host portion of the `dmdpi` debugger is written in the C++ programming language. You will need this programming language only if you will be compiling the `dmdpi` debugger source code. Binary distributions of the 630 MTG Software Development Package do not require the C++ programming language. The C++ programming language is available from AT&T Software Licensing in Greensboro, N.C. - Phone: 1-800-828-UNIX.

Storage Requirements

The 630 MTG Software Development Package source code takes over 7000 blocks of free space while the binary (executable) code takes over 5400 blocks of free space.

If you intend to compile the 630 MTG Software Development Package source code, you will need at least 25,000 blocks of free space in your file system when performing the compilation.

Location

The 630 MTG Software Development Package accepts the shell variable **DMD**, which points to the directory where the 630 MTG software is stored. The **DMD** variable lets you install the software in any directory.

The suggested installation location is */usr/opt/630*. When installing the package on a 3B2 Computer, the Simple Administration Package will automatically place the software under */usr/opt/630*. After installation is completed, the software can be moved to any directory. When installing the package on a 3B20 Computer, you have a choice of where to install the software. It is suggested that you install the software under */usr/opt/630*; however, you are not required to do so.

Time-out Feature

If your host computer uses a "time-out" feature to log off users when there is no terminal activity for a predetermined period of time, then this time-out feature must be disabled when using the 630 MTG. Many 630 MTG programs communicate with the UNIX System only when required. If you execute one of these programs while time-out is enabled, the system may log you off before execution is complete. Your System Administrator can disable the time-out feature.

Processes

The user process limit may require changing. The 630 MTG consumes processes as windows are created and programs are downloaded. When you are not creating windows or loading programs, the number of processes is usually less than 10. However, when creating windows or loading programs, the number of current processes can easily exceed 25. Therefore, tuning of the user process limit may be required.

Permission Modes

The following permission modes must be set:

- Read and execute permission on the directories from root (/) through *\$DMD/bin*
- Execute permission on the commands located in *\$DMD/bin* and *\$DMD/lib*
- Read and write permission on */dev/tty*.

Installation of 630 MTG Software Development Package

This section tells how to install the 630 MTG Software Development Package on the 3B2, 3B15, 3B4000, and 3B20 Computers. If you are installing both a binary and a source package, install the binary package before the source package.

Installation Procedure for 3B2 Computers

Installation of the binary software package for 3B2 Computers is performed through the 3B2 Simple Administration Facility. To install the binary package, place the first floppy disk into the internal floppy disk drive and type:

```
sysadm installpkg
```

The simple administration facility will guide you through the rest of the installation, and the package will be installed under */usr/opt/630*.

To install the source package, first determine a root directory where you want the software installed. If the 630 MTG binary package has previously been installed, it is recommended that the root directory of the 630 MTG source package be the same as the root directory of the 630 MTG binary package.

To complete installation of the source package:

1. Log in as *root*.
2. Place the 630 MTG Software Development Package 3B2 Source Code cartridge tape into the cartridge tape drive.
3. Change directories (**cd**) to your chosen root directory.
4. Type the following (except on 3B2/600):

```
ctccpio -iduvT /dev/rSA/ctapel
```

On a 3B2/600 use the following:

```
cpio -idcumv < /dev/rSA/qtapel
```

For instructions on compiling the source code, see the section "Building Source" in this chapter.

Installation Procedure for 3B15, 3B4000, and 3B20 Computers

Installation instructions for the binary package on a 3B15, 3B4000, or 3B20 Computer are as follows:

1. Change (`cd`) to the directory where you want to install the package (such as `/usr/opt/630`).
2. Mount the 630 MTG Software Development Package tape on a 1600 BPI (bits per inch) tape drive and type:

```
cpio -idcBumv < /dev/rmt/0m
```

Windowing Utilities Installation

The AT&T Windowing Utilities Package is a standard Utilities package delivered with UNIX System V, starting with Release 2.1.1 for the 3B20 and Release 3.0 for other AT&T processors. For earlier releases, windowing utilities are obtained from the 5620 V2.0 Core Binary package.

UNIX Systems Distributed with AT&T Windowing Utilities

If you are running a version of UNIX for which AT&T Windowing Utilities are available and they are not installed, install the AT&T Windowing Utilities Package using the UNIX System documentation that came with the package. To determine if the AT&T Windowing Utilities Package is installed, check (`ls`) to see if the file `/usr/bin/layers` exists. If it does, the utilities are already installed.

UNIX Systems Distributed without AT&T Windowing Utilities

If you are running a version of UNIX that does not have AT&T Windowing Utilities, install the 5620 Core Binary Package, including the `xt` driver, using the *5620 Dot-Mapped Display Administrator Guide* (Release 2.0) (Select Code 306-141).

For instructions on compiling the source code, see the section "Building Source" in this chapter.

Installation of 630 MTG Software Development Package ---

On the 3B2 computer, this installation will place files in the directory */usr/dmd*. On a 3B20 computer, files can be installed anywhere, with */usr/dmd* being the recommended location. The remainder of this section should be completed while logged in as **root**.

Copy the following files from the 5620 Core Binary tree into the same relative directories of the 630 MTG Software Development Package tree:

Note: Remember the suggested location of the 630 MTG Software Development Package tree is */usr/opt/630*.

- *bin/layers*
- *bin/relogin*
- *bin/xta*
- *bin/xts*
- *bin/xtt*
- *bin/ismpx*
- *bin/jwin*
- *bin/jterm*
- *bin/32ld*
- *bin/32reloc*

For example, copy:

/usr/dmd/bin/layers

to:

/usr/opt/630/bin/layers

by typing:

```
cp /usr/dmd/bin/layers /usr/opt/630/bin/layers
```

and so on....

After you have finished copying the files, set user identification (uid) to **root** for the files **layers** and **relogin**. To accomplish this, change to the directory **bin** under the 630 MTG Software Development Package tree and execute the following commands:

```
chown root layers relogin
chmod 4755 layers relogin
```

The file `$DMD/lib/hostagent.o` in the 5620 Core Package has been made into an archive file for the 630 MTG and relocated to `/usr/lib/libwindows.a`. **libwindows.a** is a library of routines which enables a program running on a host UNIX System to perform windowing terminal functions such as **New()** and **Reshape()**. To convert and relocate **hostagent.o**, execute the following command:

```
cd /usr/lib
ar cr libwindows.a /usr/dmd/lib/hostagent.o
```

The path to **hostagent.o** on the **ar** command line should be modified if the package is not installed in the `/usr/dmd` directory.

If you are on the 3B2 Computer, the **ar** command is part of the Software Generation Utilities package. If this package is not installed, skip this step. **ar** and **libwindows.a** are not usable without the Software Generation Utilities. If you later install the Software Generation Utilities package, convert and relocate **hostagent.o** at that time. Note that C language programs on the host computer will not be able to perform windowing terminal functions until **libwindows.a** is available.

A zero length file must now be created in the directory `lib/layersys` under the 630 MTG Software Development Package tree to inform the 5620 layers program of the terminal version of the 630 MTG. The name of this file must be in the form:

```
lsys.TERMINAL_VERSION
```

Installation of 630 MTG Software Development Package

Terminal version is displayed in the 630 MTG setup window. For example, if the first line of your setup window says:

8;8;6 ROMS

then your terminal version is **8;8;6**. The zero length file must be called **lsys.8;8;6**. To create the file, change to the directory *lib/layersys* under the 630 MTG Software Development Package tree and type:

```
> "lsys.8;8;6"
```

If multiple 630 MTG terminals are used on your computer and the terminals have different firmware versions, more than one file will have to be created. Some zero length files already exist in this directory. If the zero length file for a particular firmware version is missing, the following error message will be displayed when the person using the terminal with that firmware version executes the **layers** command:

```
5620 Software - Firmware mismatch. . .
```

If no 5620 terminals will be used on the system, files under the root directory of the 5620 may now be removed, if desired, in order to recover disk space.

If you installed the binary 630 MTG Software Development Package, installation is now complete.

Building Source Code

When installing source code, it is recommended that you also install the appropriate binary package. If you do not install the binary package, you will have to compile all the source code.

The host portion of the **dmdpi** debugger is written in the C++ programming language. C++ is available from AT&T Software Licensing in Greensboro, N.C. - Phone: 1-800-828-UNIX. Before compiling the 630 MTG Software Development Package from source, C++ must be installed on your computer. See the file *\$DMD/src/dmdpi/README* for additional information about compiling **dmdpi**.

Installation of 630 MTG Software Development Package

To build the entire 630 MTG Software Development Package:

1. Your environment should be set up for a 630 MTG (described under "User's .profile" in the "Overview" Chapter of this document).
2. Verify that there are at least 25,000 blocks of free space in the file system that contains **\$DMD**.
3. Execute the following commands:

```
cd $DMD  
make ACTION=install all
```

Note: The newly compiled software will be installed in the directories *\$DMD/bin* and *\$DMD/lib*.

If you want to free up disk space after the package has been built, use the following command to remove temporary files from your source directories:

```
make ACTION=clobber all
```

To build manual pages for the 630 MTG Software Development Package:

```
cd $DMD/man/src  
makeall
```

Note: You must have DOCUMENTER'S WORKBENCH* software installed on your host computer to build manual pages.

* Trademark of AT&T.

Local Software

Partitioning of local and official software is recommended to ensure proper software support. The executable version of locally developed software should be placed in *\$DMD/local/bin* and have execute permission for 630 MTG users. Users of locally developed software will have to add this directory to their *\$PATH* variable.

Source for locally developed software should be stored in *\$DMD/local/src*, with manual pages stored in *\$DMD/local/doc*.

Appendix B: 5620 Compatibility

Co-existence of 5620 DMD and 630 MTG on the Same Host Computer	B-1
Porting 5620 DMD Programs to the 630 MTG	B-3
Change in Processors - Different Word Size	B-3
The File "5620.h"	B-6
When to Use "5620.h"	B-6
How to Use "5620.h"	B-6
What "5620.h" Fixes	B-7
Assorted Problems Not Fixed by Including "5620.h"	B-10
Host-Related Changes	B-12
"New" and "Reshape"	B-12
Using Identical Source Files on Both Terminals	B-12

Co-existence of 5620 DMD and 630 MTG on the Same Host Computer

When the 5620 DMD and 630 MTG terminals and associated software packages co-exist on the same host computer, users who need the ability to use both types of terminals must set their **PATH** variables to the appropriate software tree, depending on the type terminal being used. This section provides two simple ways to set up the user's environment that will ensure the correct software package is accessed.

The following sample *.profile* will ask a user for the terminal type and set up an appropriate environment for either a 630 MTG or 5620 DMD based upon the user's response:

```
echo Terminal? \\c
read TERM
case $TERM in
630)
    stty tabs ixon erase \\^h
    DMD=/usr/opt/630 # or wherever the 630 MTG software is located
    PATH=$PATH:$DMD/bin
    export TERM DMD PATH
    ;;
5620)
    stty tabs ixon erase \\^h
    DMD=/usr/dmd # or wherever the 5620 DMD software is located
    PATH=$PATH:$DMD/bin
    export TERM DMD PATH
    ;;
esac
```

Co-existence of 5620 DMD and 630 MTG on the Same Host Computer —

Alternatively, a shell program similar to the following can be used to switch between terminal environments. This program assumes that the two variables, `$DMD5620` and `$DMD630`, are set in the user's *.profile*. These variables should point to the 5620 DMD and 630 MTG software trees, respectively. When executed, this program switches environments between 5620 DMD and 630 MTG depending on the terminal being used.

This program can be typed into a file and then executed when a user switches between terminals.

```
if [ "$DMD5620" = "" -o "$DMD630" = "" ]
then
    echo "\$DMD5620 and/or \$DMD630 not set"
elif [ "$DMD" = "$DMD630" ]
then
    PATH='echo $PATH | sed "s'$DMD630'$DMD5620'g"'
    DMD=$DMD5620
    TERM=5620
    echo "Now set for 5620"
else
    PATH='echo $PATH | sed "s'$DMD5620'$DMD630'g"'
    DMD=$DMD630
    TERM=630
    echo "Now set for 630"
fi
export DMD PATH TERM
```

Note: This program must be executed in the shell process by typing the command as follows: `. file`

Porting 5620 DMD Programs to the 630 MTG

This section describes several areas that should be considered when porting 5620 DMD 'C' language programs for execution in the 630 MTG environment.

Note: This section is primarily intended for users who are porting programs to the 630 MTG. If more information is needed on the 5620 DMD, consult the appropriate 5620 DMD documentation.

Although the execution environments for the 5620 DMD and 630 MTG are basically compatible, some programming changes are necessary due to the fact that the 5620 DMD and 630 MTG have different processors. The effects on 'C' language programs, as a result of this difference, are explained in this section.

The 630 MTG Software Development Package provides the include file **5620.h** that resolves many, but not all, of the discrepancies between the two execution environments. This file may be included in all 5620 DMD programs ported to the 630 MTG.

This section covers the following areas:

- Porting difficulties that are inherent with the change in processors
- How to use **5620.h** and a description of problems that can be solved by including **5620.h** in the source file
- Descriptions of other potential problems (not resolved in **5620.h**) and ways to solve them
- Host agent differences between the 5620 DMD and 630 MTG
- How to use the same source file for both terminals.

Change in Processors - Different Word Size

In C language, the size of an integer is defined as the word size of the machine. This is a portability issue for any 'C' code and is generally solved by not assuming word size. For example, always defining variables as either short or long instead of **int** can prevent this from becoming a problem in porting code.

Porting 5620 DMD Programs to the 630 MTG

The 5620 DMD uses a WE*32100 microprocessor, which has a 32-bit data bus. The size of a word, then, is defined as 4 bytes. With the Motorola MC68000† and its 16-bit data bus, a word is only 2 bytes.

Problems that can result from incorrectly assuming an integer is 32 bits on the 630 MTG are:

- A. **Value Does Not Fit in 16 Bits**—The integer may take on a value larger than can be represented with only 16 bits. If an integer is expected to take on large values, it should be changed to a long.
- B. **Parameter Passing Errors**—Sometimes an integer is passed where a long or a pointer is expected. Integers are now the size of a short, not of a long or pointer. Passing an integer can have unpredictable results.

Routines that accept an integer on the 5620 DMD but require a long on the 630 MTG are: **itox**, **sqrt**, and **galloc**. (**sqrt** has been renamed **lsqrt** on the 630 MTG.) See the manual page **ITOX(3L)**, **LSQRT(3L)**, and **GALLOC(3R)** in the *630 MTG Software Reference Manual* for more details.

For example, for the **itox** routine declared as:

```
char *itox(i, s)
long i;
char *s;
```

The call:

```
itox(10, &buf);
```

should be written as:

```
itox((long)10, &buf);
```

The reverse case of using a long or pointer in place of an integer is less common though still a problem. For example, in **printf**, passing a

* Registered trademark of AT&T

† Trademark of Motorola, Inc.

pointer to be displayed as an integer will display only the first two bytes and will alter the remaining arguments.

This problem can be corrected by using a cast, declaring the integer as a long, or changing the type of the parameter.

- C. **Relying on a Structure Being a Certain Size**—Any structure that had a field defined as an integer has changed size. Also, structures are now rounded to the nearest 2 instead of 4 bytes. This should not affect programs that used the **'sizeof'** operator to communicate the size of a structure.

The following data types and structures have changed sizes between the 5620 DMD and the 630 MTG.

sizeof()	5620	630
int	4	2
Word	4	2
Code	4	2
Bitmap	20	18
Mouse	12	10
Fontchar	8	6
Font	20	18

- D. **Change in Bitmap "width"**—A common problem with porting 5620 DMD code to the 630 MTG occurs with the declaration of Bitmaps. Because of the change in the value of **sizeof(Word)**, the **width** field of the Bitmap must be doubled. To make Bitmaps compatible with both the 5620 DMD and 630 MTG, the following guidelines should be followed:

1. Declare the image data as an array of unsigned longs or unsigned shorts rather than an array of Words since the size of Word changes for different processors.
2. As a result of the previous step, you will need to cast **base** in the Bitmap data structure as a pointer to a Word.

3. Declare the **width** field in the Bitmap data structure as follows:

```
N/sizeof(Word);
```

where **N** is the number of bytes in a single scan-line of the Bitmap's image data. **N** must be a multiple of **sizeof(Word)**.

The File “5620.h”

When to Use “5620.h”

The **5620.h** file is useful for porting code from the 5620 DMD to the 630 MTG. New programs written for the 630 MTG do not need to include **5620.h** in their source code. **5620.h** mostly contains definitions of 5620 DMD functions in terms of their 630 MTG counterparts. Including **5620.h** can save you some editing of your source files. The **5620.h** file is located in directory *\$DMD/include*.

How to Use “5620.h”

If you are trying to compile a 5620 DMD source file for the 630 MTG, you can edit the file to add the line:

```
#include <5620.h>
```

5620.h should be included AFTER **dmd.h**, since it redefines a few items in **dmd.h**.

If you have a large number of source code files, you may wish to make a local copy of the include file **dmd.h** and have it include **5620.h**. It is not recommended that you edit the **dmd.h** in *\$DMD/include*. Make your own copy to ensure proper software support of your official 630 MTG software.

If compatibility with the 5620 DMD is not an issue once your programs are running on the 630 MTG, it is recommended that you remove the dependency, if any, on **5620.h** after the initial port. At some time, edit the 630 MTG file to take better advantage of 630 MTG features and to avoid extra macro calls.

What "5620.h" Fixes

This section describes potential porting problems that are resolved by including **5620.h**.

- A. **A Few Function Name Changes**—Because the 630 MTG has both floating point and integer approximation routines, there are name conflicts with the integer approximation routines of the 5620 DMD. On the 630 MTG, the "I" and "L" in the name indicate integer and long to differentiate from the regular floating point routines. This is shown in the table.

OLD NAME	NEW NAME
sin	Isin
cos	Icos
atan2	Iatan2
floor	Ifloor
ceil	Iceil
sqrt	Lsqrt

Also, the functions **outline** and **getrect** have been replaced with the more general functions **box** and **newrect**. **5620.h** resolves these changes by defining macros that do the name translation. For example, **outline** is defined as:

```
#define outline(olr)    box(&physical, olr, F_XOR)
```

Finally, the 5620 DMD function **texture** is not supported on the 630 MTG, and the 5620 DMD function **texture16** is called **texture** in the 630 MTG. The include file **5620.h** performs this translation for programs ported to the 630 MTG.

- B. **New Method of Linking Firmware Functions**—On the 5620 DMD, firmware routines are accessed by a macro definition that casts an absolute address to a pointer, to a function, and then calls that function. This is a burden on **cpp** and creates error messages when the address of the function is used or a variable or function is declared with the same name as that of a firmware function.

The new method of accessing firmware routines is done by linkages, similar to the way UNIX System calls are done. When the program is linked together, the library **libfw** is searched to satisfy undefined external functions. This library holds small (3 lines) assembly language routines that call firmware routines via their addresses stored in a vector table. This has the following effects on source files:

- On the 5620 DMD, a function with the same name as one in the firmware has to be undefined first in every file referencing it. This is unnecessary but harmless on the 630 MTG.
- Because of this new linkage, firmware functions that do not return an integer must be defined as external to get the correct return type. **5620.h** does this for documented 5620 DMD routines.

C. **A Few New Macro Definitions**—The macros **Pt**, **Rpt**, and **Rect** have new definitions in **dmd.h**. The new macros are declared as:

```
#define Pt(x,y)          x, y
#define Rpt(x,y)        x, y
#define Rect(a,b,c,d)   a, b, c, d
```

This works because of the way the 68000 C Compilation System (CCS) passes a structure, and it is more efficient than calling another routine to construct the structure. **5620.h** redefines these macros as calls to the routines **fPt**, **fRpt**, and **fRect**. These routines are functionally equivalent to the 5620 DMD's routines which were recommended for parameter passing.

D. **No Defont Font**—There is no longer a font called **defont**. Use **mediumfont** instead. In **5620.h**, **defont** is defined as **mediumfont**.

- E. **Certain Font Definitions No Longer Meaningful**—In the 5620 DMD include file **setup.h**, some font properties are listed as:

```
#define CW      9      /* width of a character */
#define NS      14     /* newline size;        */
                        /* height of a character */

#define CURSOR  "\1"

#define XMARGIN 3      /* inset from border   */
#define YMARGIN 3
#define XCMAX ((XMAX-2*XMARGIN)/CW-1)
#define YCMAX ((YMAX-2*YMARGIN)/NS-3)
```

These are still defined, but as a part of **5620.h**. On the 630 MTG, use the macros **FONTWIDTH** and **FONTHEIGHT** to find the width and height of characters in a given font (documented in the *630 MTG Software Reference Manual* under **string(3R)**). The old **CW**, **NS**, **XCMAX**, and **YCMAX** are only applicable to **mediumfont** on the 630 MTG. The problem with the **CURSOR** definition is that ascii "\1" is not a cursor in **mediumfont** as it was in **defont**. The function **rectf** will have to be used to create a rectangular cursor.

- F. **"MPX" Defined in "5620.h"**—Since there is no **stand-alone** environment on the 630 MTG (see below), **MPX** should always be defined for 5620 DMD applications. This is set in **5620.h**.

Assorted Problems Not Fixed by Including “5620.h”

There are other differences between the application environments of the 5620 DMD and the 630 MTG that can potentially cause porting problems. Some will only affect a small number of 5620 DMD programs. Read through this section to see if your program is affected. Code changes will be necessary to resolve these issues; they are not resolved in the include file **5620.h**.

- A. **No “Stand-Alone” Environment**—If you have a program written for the **stand-alone** programming environment of the 5620 DMD, you will need to port it to the **layers** environment before trying to run it on the 630 MTG. The 630 MTG does not support **stand-alone**; a program written for the **layers** environment will run in **non-layers**, although it must compensate for the lack of an error-correcting protocol if it does so. This is why **MPX** is defined in **5620.h**.
- B. **Names of Include Files**—The 5620 DMD Application Development Package has some include files that are linked together. This is not done in the 630 MTG Software Development Package. Some of the include files have changed names. These are shown in the following table.

OLD NAME	NEW NAME
blit.h	dmd.h
blitio.h	dmdio.h
jerq.h	dmd.h
jerqio.h	dmdio.h
jerqproc.h	dmdproc.h

Also, **pandora.h** is no longer available; it defined macros for calling unsupported firmware functions. These values in the old file are not valid for the 630 MTG. However, many of the routines are now supported and documented in the *630 MTG Software Reference Manual*.

The files **duart.h**, **kbd.h**, **line.h**, **queue.h**, **sa.h**, **sys/2681.h**, **sys/xt.h**, **sys/jioctl.h**, and **sys/xtproto.h** in the 5620 DMD’s **include** directory do not exist for the 630 MTG because they are hardware specific or not applicable to the 630 MTG environment.

- C. **Textures**—The 630 MTG does not support a 32-bit **Texture** structure. The 5620 DMD supports both **Texture** and **Texture16**. The 630 MTG has only one: **Texture16**. On both terminals, the **Texture16** is a 16 by 16 bit array, so those are completely compatible. The declaration of **Textures** in programs will have to be changed. Often a 32 by 32 bit **Texture** is a small pattern repeated over and over, and reducing the size saves space without losing any detail. (See **texture(3R)** in the *630 MTG Software Reference Manual*.)
- D. **Sign Extension of Characters**—The 630 MTG's C Compilation System (CCS) does sign extension on characters. For example, if a **char** is assigned to an **int**, the sign bit will be propagated. This is not true in the 5620 DMD's Software Generation System (SGS). A way to handle this is to use **unsigned char** rather than just **char**.
- E. **Graphical Displays of Memory Usage**—Since the memory allocation scheme in the 630 MTG is different from that in the 5620 DMD, programs which display memory usage will have to be rewritten. The 630 MTG program **dmdmemory** is a good example of how to display 630 MTG memory usage.
- F. **"Hostagent.o" is now "libwindows.a"**—The file `$DMD/lib/hostagent.o` in the 5620 DMD Core Package is made into the library `/usr/lib/libwindows.a` by the system administrator when the 630 MTG software is installed on systems with UNIX System V Release 2. (This procedure is described in the "Windowing Utilities Installation" section of Appendix A.) This change affects programs that use **hostagent.o**, since they must now include **-lwindows** on the compile line or in their makefile.
- G. **PF Keys**—The PF keys have the values 0x80 to 0x97 on the 630 MTG in NOPFEXPAND mode. On the 5620 DMD, they have the values 0x82 to 0x89. See **keyboard(3R)** in the *630 MTG Software Reference Manual* for more information about NOPFEXPAND mode.

Host-Related Changes

“New” and “Reshape”

The host agents **New** and **Reshape** windowing utilities commands have changed such that the exterior includes the window's border and the interior does not. For example, when called with the rectangle (8,8,264,264), the resulting window will be:

TERMINAL	Exterior	Interior
5620 v1.1	(8,8,264,264)	(10,10,262,262)
5620 v2.0	(6,6,266,266)	(10,10,262,262)
630	(8,8,264,264)	(12,12,260,260)

Both routines will also now clip the rectangle to the screen.

Using Identical Source Files on Both Terminals

It is possible to use the same source file in both the 5620 DMD and 630 MTG programming environments. However, the parts of the source code specific to a terminal or programming environment need to be indicated when the file is compiled.

For the 630 MTG, the variable **DMD630** is automatically defined when a source file is compiled with **dmdcc**. The environment specific code can then be enclosed in **#ifdef DMD630...#endif** directives (or, alternatively, **#ifndef DMD630...#endif** for 5620 DMD code).

As an example, if all of your porting problems can be fixed by **5620.h**, you can add the following lines to your file and the code would compile correctly for both environments.

```
#ifndef DMD630
#include <5620.h>
#endif
```

Index

5620 Compatibility,*B-1*
630 MTG Documentation,*1-2*
630 MTG font cache,*8-12*
630 MTG operating system,*3-2*
630 MTG Software Development
Package
 features,*1-5*
 introduction,*1-1*

A

alarm,*5-14*
ALARM resource,*5-14*
application caching facility,*10-1*
application resource management,
 5-1
applications,
 all users,*1-5*
 programmers,*1-6*
archive,*15-5*
archive file member header,*15-5*
archive headers,*15-5*
archive symbol table,*15-5*
argument,
 argc,*2-8*
 argv,*2-8*
assembler window,*12-29*
attach,*5-11*

B

bfm demonstration - Tmenu9.c,*6-14*
bitblt,*4-32*
 illustrations,*4-33*
Bitmap,*4-5*
Bitmap illustrations,*4-6*
breakpoints - clearing,*12-23*

breakpoints - setting,*12-13*
bttns,*5-6*
button interface function - "bttns",-
 5-6
button interface macros,*5-5*

C

C Compilation System (CCS),*15-1*
C language,*15-16*
C++ Programming Language,*A-1*
cache routine,*10-2*
cached application - removing,*10-8*
cached application - reshapability,-
 10-9
cached application - restarting,*10-5*
cached application - .text, .data,
 .bss,*10-11*
caching an application,*10-2*
caching,
 host-connected application,*10-6*
 local or connected application,-
 10-6
 shared application,*10-6*
call stacks,*12-15*
CCS utilities,*15-1*
chapter descriptions,*1-7*
co-existence - 5620 DMD and 630
 MTG,*B-1*
co-existence - PATH variable,*B-1*
co-existence - user's .profile,*B-1*
Comparisons,*4-15*
compiling source code,*A-8*
conditional breakpoints,*12-27*
Conversions,*4-15*
coordinate system transformations,-
 4-25

- coordinate systems,4-21
- copies of applications,5-12
- CPU resource,5-13
- CPU Sharing,3-4
- creating an archive,15-5
- current point,4-23
 - screen coordinate system,4-24
 - window coordinate system,4-24
- current screen point,2-7
- cursallow,5-6
- cursinhibit,5-6
- cursor keys,11-3
- curset,5-7
- cursswitch,5-7

D

- data type conversions,4-19
- data types,4-2
- data types - Menu,6-2
- data types,
 - Bitmap,4-5
 - Point,4-2
 - Rectangle,4-2
 - Word,4-4
- debugging crashed processes,12-33
- DELETE resource,5-13
- dfn demonstration - Tmenu9.c,6-14
- displaying icons,6-16
- displaying text,6-17
- displaying text and icons,6-17
- DMD variable,A-2
- dmda.out file,15-3
- dmdcc command,15-3
- dmdcc command options,15-4
- dmddemo,1-5
- dmdman,1-5
- dmdpi window,12-8

- dmdpi,
 - demonstration,12-6
 - description,12-1
 - help window,12-5
 - keyboard input,12-4
 - mouse operation,12-2
 - other features,12-25
 - special cursor icons,12-4
 - user interface,12-2
 - working directory,12-31
- documentation,1-2
 - Reference Manual,1-2
 - Release Notes,1-2
 - User's Guide,1-2
- drawing characters,8-5
- drawing routines,4-27
 - screen coordinates,4-27
 - window coordinates,4-29
- drawing text strings,8-8
- drawing with the mouse,5-7

E

- example of caching,10-4
- Example Program,
 - arguments.c,2-8
 - cache10.c,10-34
 - cache11.c,10-36
 - cache12.c,10-38
 - cache13.c,10-40
 - cache14.c,10-42
 - cache15.c,10-44
 - cache16.c,10-47
 - cache17.c,10-48
 - cache1.c,10-21
 - cache2.c,10-22
 - cache3.c,10-23
 - cache4.c,10-24

Example Program (Continued)

cache5.c,10-25
 cache6.c,10-26
 cache7.c,10-28
 cache8.c,10-31
 cache9.c,10-32
 caching a downloaded font,-
 8-14
 clock.c,12-35
 compiling - dmdcc,2-3
 downloading - dmdld,2-4
 drawchar,8-6
 general,2-1
 getfont,8-10
 hello.c,2-3
 HELLO.c,2-5
 jxmouse,7-6
 kbd1.c,11-11
 label1.c,6-67
 label2.c,6-68
 label3.c,6-69
 Menu1.c,6-21
 Menu2.c,6-23
 messages1.c,9-8
 messages2.c,9-14
 mouse.c,5-17
 msgbox1.c,6-71
 msgbox2.c,6-73
 screen.c,4-43
 star.c,5-29
 Tmenu1.c,6-27
 Tmenu2.c,6-30
 Tmenu3.c,6-33
 Tmenu4.c,6-37
 Tmenu5.c,6-42
 Tmenu6.c,6-47
 Tmenu7.c,6-52
 Tmenu8.c,6-59

Example Program (Continued)

Tmenu9.c,6-64
 TrackMouse.c,5-27
 twist.c,4-41
 type.c,5-35
 vsterm1.c,5-37
 exceptions,3-5
 explicit arguments - "s" and "f",10-2

F

features,
 software package,1-5
 font cache,8-12
 Font data structure,8-1
 font field - initialization,6-12
 Fontchar data structure,8-3
 fonts from the host,8-9
 function codes - graphics routines,-
 4-26
 function keys,11-2

G

getfont,8-9
 Getting Started,
 materials needed,2-2
 porting 5620 programs,2-2
 what you need to know,2-1
 global structure "mouse",5-4
 global structures - coordinate
 systems,4-21
 global variable "P",3-2
 global variables - inspect,12-22
 global variables, eliminating,10-14
 Globals window,
 keyboard expressions,12-25
 graphic routine built-in mouse
 interface,5-7

graphical coordinate systems,4-21
graphical routines,4-26
greying menu items - Tmenu6.c,6-13

H

hfn demonstration - Tmenu9.c,6-14
host communications,5-10
hostagent.o file,A-7
host-related changes,B-12

I

icon commands,14-6
icon editor,14-1
 arrow command,14-9
 arrow menu,14-9
 background grid command,-
 14-13
 bitblt command,14-13
 copy command,14-10
 drawing with,14-9
 duplicate command,14-12
 erase command,14-10
 exit command,14-15
 invert command,14-10
 mouse command,14-13
 read file command,14-12
 Reflect X command,14-11
 Reflect y command,14-11
 Rotate + command,14-11
 Rotate - command,14-11
 Shear X command,14-11
 shear y command,14-11
 stretch command,14-12
 Texture16 command,14-12
 write file command,14-15
icon menu,14-5

icons,
 initiating an icon session,14-3
 selecting bitmap or texture16,-
 14-4
 storing,14-2
 supplied,14-18
 two data types,14-2
 use in programs,14-15
 using,14-3

Image of the world,2-7
implicit arguments,10-3
include file - world.h,4-13
include file 5620.h,B-6
inclusion operations,4-19
initializing the .bss section,10-13
installation and administration,A-1
installing software,A-4
installing windowing utilities,A-5
interprocess communications,9-1

J

jim editor,13-1
 adding new text,13-8
 button 3 menu,13-15
 changing text,13-9
 command summary,13-24
 commands,13-13
 copying text,13-8
 diagnostic messages,13-17
 file loading,13-5
 frame positioning,13-14
 how to quit,13-17
 moving text,13-8
 naming files,13-11
 operations,13-4
 recovering lost files,13-22
 save buffer,13-10

jim editor *(Continued)*
 scrolling,13-14
 special characters,13-15
 string searches,13-12
 text deletion,13-7
 text selection,13-6
 using mouse,13-4
 work frame,13-4
 writing files,13-10
 jim window,13-2
 Journal,12-28
 jstring,8-8
 jx,
 functions,7-4
 how it works,7-2
 I/O interpreter,7-1
 using,7-3

K

kbdchar,11-1, 5-8
 key clusters,11-2
 keyboard and printer as a
 typewriter,5-9
 keyboard expressions,12-25
 keyboard modes - "kbd2.c",11-9
 keyboard redefinition,
 demonstration,11-9
 keyboard resource,5-8
 keyboard,
 cursor keys,11-3
 entire keyboard,11-6
 function keys,11-2
 identification,11-7
 key clusters,11-2
 key positions,11-26
 layout,11-25
 NOTRANSLATE,11-6

keyboard *(Continued)*
 numeric key pad,11-4
 operation,11-6
 redefining,11-1
 scroll lock key,11-5
 transmittal codes,11-22
 keystrokes,11-7

L

label bar,6-15
 label bar - requesting,6-16
 labelicon,6-16
 labeloff,6-16
 labelon,6-16
 labeltext,6-17
 lib/layersys directory,A-7
 library files,15-5
 link editor,15-5
 linking the menus,6-11
 loadfont,1-5
 local,5-11
 local software,A-10
 local variables - inspect,12-17
 lprintf,2-4, 2-7

M

maintaining text and icons,6-18
 mc68ar command,15-5
 MC68AS,16-1
 mc68as command,
 format,16-4
 MC68AS sub instruction,16-3
 MC68AS,
 address mode syntax,16-21
 comments,16-6
 constants,16-7

MC68AS (*Continued*)
 expressions,16-12
 identifiers,16-6
 line format,16-5
 machine instructions,16-24
 pseudo-operations,16-13
 register identifiers,16-7
 segments, location counters,
 labels,16-9
 span-dependent optimization,-
 16-19
 syntax rules,16-5
 types,16-11
mc68as,
 use of,16-4
MC68AS,
 warnings,16-2
mc68conv command,15-6
mc68cprs command,15-7
mc68dis command,15-7
mc68dump command,15-8
mc68ld command,15-9
mc68lorder command,15-9
mc68nm command,15-10
mc68size command,15-11
mc68strip command,15-11
Menu,6-2
menu expansion - Tmenu3.c,6-11
menu item bitmaps - Tmenu4.c,6-12
menu, simple - Menu1.c,6-5
menu, simple - Tmenu1.c,6-10
menuhit,6-2
 use of,6-5
message boxes,6-19
message,
 creating,9-3
 definition,9-2
 queue control,9-11

message (*Continued*)
 receiving,9-6
messages,9-1
message,
 sending,9-5
modifying the .data section,10-12
mouse cursor control,5-6
mouse interaction,6-4
mouse resource,5-4
mouse tracking,5-5
MSG resource,5-15
msgget,9-3
msgrcv,9-6
msgsnd,9-5
msqid,9-6
msqid_ds,9-11
multiple fonts - Tmenu5.c,6-12
multiple menus - Menu2.c,6-5

N

newrect,5-7
next field - Tmenu2.c,6-11
NOTRANSULATE demonstration -
 "kbd3.c",11-9
NOTRANSULATE protocol,11-6
numeric key pad,11-4

O

operating system,3-2
 processes,3-2
Operations,4-15
operations on Points,4-15
organization,1-7
own,5-2

P

P->appl,10-16
 parameter passing,10-15
 parameter passing - "P->appl",10-16
 peel,5-12
 permission modes,A-3
 Point,4-2
 Point comparison,4-16
 Point operations,4-15
 porting 5620 DMD programs,B-3
 printer and keyboard as a
 typewriter,5-9
 printer resource,5-9
 printing, dmdcat,1-5
 printing text,
 lprintf,2-4
 process control window,12-9
 process exceptions,3-5
 process inhibit - lockup,3-4
 process scheduling,3-3
 process states,3-3
 process structure,3-2
 process switch,3-4
 process switch notification,11-8
 processes,3-2
 profile, user,1-3
 psendchar,5-9
 psendchars,5-9
 "P", warnings,10-17

R

rcvchar,5-10
 receiving and sending data,5-10
 Rectangle,4-2
 Rectangle comparison,4-18

Rectangle operations,4-17
 regaining the host connection,5-11
 register use,15-12
 releasing the host connection,5-11
 request,5-1
 requesting a resource - "request",5-1
 reshapability - cached application,-
 10-9
 RESHAPED resource,5-14
 resources command,
 own,5-2
 request,5-1
 wait,5-3
 Resources,
 ALARM,5-14
 CPU,5-13
 DELETE,5-13
 keyboard,5-8
 mouse,5-4
 MSG,5-15
 printer,5-9
 RESHAPED,5-14
 return value of cache,10-4
 ringbell,5-8

S

save buffer,13-10
 saving the .bss section,10-13
 scroll lock key,11-5
 sections - .text, .data, .bss,10-11
 sendchar,5-10
 sending data to the printer,5-9
 servicing a resource - "own",5-2
 setting the mouse position,5-7
 shared text applications,
 writing,10-14
 sharing the .bss section,10-12

sharing the CPU,3-4
sharing the CPU - "wait(CPU)",2-5
sharing the .data section,10-12
software dependencies,A-1
software installation,A-4
software package directory,A-2
source files - 5620 DMD/630MTG,-
 B-12
source window,12-10
Spy,12-28
stack frame,
 keyboard expressions,12-25
stack frames,12-15
starting a jim session,13-4
static menus - Tmenu7.c,6-13
static menus and multiple selections
 - Tmenu8.c,6-14
stderr,7-2
stdin,7-2
stdout,7-2
step statements,12-20
storage requirements,A-2
string,8-8
structure of label bar,6-15
strwidth,8-8
subroutine calls - Tmenu6.c,6-13
syntax rules,16-5
system exceptions,3-5
system services,5-13

T

terminal emulator - "vstern1.c",5-11
Textures - data type,4-30
time-out feature,A-2
Titem definition,6-8
Tmenu definition,6-7
tmenuhit - tree menus,6-6

tmenuhit data types - Tmenu and
 Titem,6-6
tmenuhit,
 calling,6-9
 return value,6-10
 use of,6-10
tools - user interface,6-1
turning the mouse cursor Off and
 On,5-6

U

user interface tools,6-1
user process limit,A-3
user responsibilities,1-3
 install software,1-3
 profile,1-3
using different mouse cursors,5-7

W

wait,5-3
wait(CPU),2-5
waiting on a resource - "wait",5-3
windowing utilities,A-1
Word,4-4
word size,B-3
writing shared text applications,-
 10-14
 eliminating global variables,-
 10-14
 modifying global variables,-
 10-14
 porting existing applications,-
 10-18