



**iRMX™ 86
OPERATOR'S MANUAL**

**iRMX™
86
OPERATING
SYSTEM**

**iRMX™ 86
OPERATOR'S MANUAL**

Order Number: 144523-001

REV.	REVISION HISTORY	PRINT DATE
-001	Original Issue. Contains information formerly found in iRMX 86 HUMAN INTERFACE REFERENCE MANUAL and iRMX 86 INSTALLATION GUIDE. This issue documents Release 5 of the iRMX 86 Operating System.	11/82

Additional copies of this manual or other Intel literature may be obtained from:

Literature Department
Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

The information in this document is subject to change without notice.

Intel Corporation makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Intel Corporation assumes no responsibility for any errors that may appear in this document. Intel Corporation makes no commitment to update nor to keep current the information contained in this document.

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

Intel software products are copyrighted by and shall remain the property of Intel Corporation. Use, duplication or disclosure is subject to restrictions stated in Intel's software license, or as defined in ASPR 7-104.9(a) (9).

No part of this document may be copied or reproduced in any form or by any means without prior written consent of Intel Corporation.

The following are trademarks of Intel Corporation and its affiliates and may be used only to identify Intel products:

BCP	Insite	iPDS	Micromap
CREDIT	Intel	iRMX	Multibus
i	Intel	iSBC	Multichannel
P-ICE	Intelelevision	iSBX	Multimodule
ICE	Intellec	iSXM	Plug-A-Bubble
iCS	intelligent Identifier	Library Manager	PROMPT
iLBX	intelligent Programming	MCS	RMX/80
im	Intellink	Megachassis	RUPI
iMMX	iOSP	Micromainframe	System 2000
			UPI

PREFACE

This manual is the primary reference for operators who access the iRMX 86 Operating System through a terminal using Human Interface commands. In addition to Human Interface commands, this manual also discusses the line-editing and control characters supported by the iRMX 86 Operating System and two utilities available to iRMX 86 operators: the Patching Utility and the Files Utility.

The manual is divided into the following chapters:

Chapter 1 discusses the Line-editing and control characters available to terminals that access the iRMX 86 Operating System. This chapter applies to all application systems, regardless of whether they include the Human Interface.

Chapters 2 through 4 discuss the Human Interface. Chapter 2 introduces the operator to the Human Interface and describes the general process of using it. Chapter 3 provides a detailed description of Human Interface commands in alphabetical order. Chapter 4 contains examples of Human Interface operations.

Chapter 5 discusses the Patching Utility, a utility that runs on both iRMX 86-based systems and Series III Microcomputer Development Systems.

Chapter 6 discusses the Files Utility, an iRMX 86 application system that you can use to format and maintain iRMX 86 secondary storage volumes.

Appendix A contains a list of iRMX 86 condition codes with short descriptions of the codes.

NOTATIONAL CONVENTIONS

This manual uses the following notational conventions to illustrate syntax.

<u>UPPERCASE</u>	In examples of command syntax, uppercase information must be entered exactly as shown. You can, however, enter this information in uppercase or lowercase characters.
<u>lowercase</u>	In examples of command syntax, lowercase fields indicate information to be supplied by the user. You must enter the appropriate value or symbol for lowercase fields.
<u>underscore</u>	In examples of dialog at the terminal, user input is underscored to distinguish it from system output.

PREFACE (continued)

- ◁ Angle brackets surround variable fields in messages displayed by the Human Interface commands and by the utilities. This information can vary from message to message.

All numbers, unless otherwise noted, are assumed to be decimal. Hexadecimal numbers include the "h" radix character (for example OFFh).

RELATED PUBLICATIONS

The following manuals provide additional background and reference information:

- Introduction to the iRMX™ 86 Operating System, Order Number: 9803124
- iRMX™ 86 Nucleus Reference Manual, Order Number: 9803122
- iRMX™ 86 Terminal Handler Reference Manual, Order Number: 143324
- iRMX™ 86 Basic I/O System Reference Manual, Order Number: 9803123
- iRMX™ 86 Extended I/O System Reference Manual, Order Number: 143308
- iRMX™ 86 Loader Reference Manual, Order Number: 143318
- iRMX™ 86 Human Interface Reference Manual, Order Number: 9803202
- iRMX™ 86 Configuration Guide, Order Number: 9803126
- iRMX™ 86 Programming Techniques, Order Number: 142982
- iRMX™ 86 Disk Verification Utility Reference Manual, Order Number: 144133
- iAPX 86,88 Family Utilities Users' Guide, Order Number: 121616
- User's Guide for the iSBC® 957B iAPX 86,88 Interface and Execution Package, Order Number: 143979
- Guide to Writing Device Drivers for the iRMX™ 86 and iRMX™ 88 I/O Systems, Order Number: 142926
- iMMX™ 800 Multibus® Message Exchange Reference Manual, Order Number: 144912

CONTENTS

	PAGE
CHAPTER 1	
LINE EDITING AND CONTROL CHARACTERS	
Type-Ahead.....	1-1
Controlling Input to a Terminal.....	1-2
Controlling Output to a Terminal.....	1-3
Escape Sequences.....	1-5
CHAPTER 2	
USING THE HUMAN INTERFACE	
Requirements.....	2-1
Configurable Features of the Human Interface.....	2-1
Loading the Operating System.....	2-2
Accessing the Human Interface.....	2-4
File Structure.....	2-5
Types of Files.....	2-5
Named File Hierarchy.....	2-6
Pathnames.....	2-8
Logical Names.....	2-9
Logical Names for Devices.....	2-10
Logical Names for Files.....	2-11
Where Logical Names are Stored.....	2-11
Logical Names Created by the Operating System.....	2-13
Removing Volumes from Devices.....	2-13
Wild Cards.....	2-14
Command Syntax.....	2-17
Command Name.....	2-19
Prepositions.....	2-20
Inpath-List and Outpath-List.....	2-21
One-for-One Match.....	2-22
Concatenate.....	2-22
Error Conditions.....	2-23
Other Parameters.....	2-23
System Manager.....	2-24
CHAPTER 3	
HUMAN INTERFACE COMMANDS	
Error Messages.....	3-1
Command Syntax Schematics.....	3-3
ATTACHDEVICE.....	3-7
ATTACHFILE.....	3-13
BACKUP.....	3-16
COPY.....	3-24
CREATEDIR.....	3-28
DATE.....	3-29
DEBUG.....	3-31
DELETE.....	3-33
DETACHDEVICE.....	3-35
DETACHFILE.....	3-38

CONTENTS (continued)

	PAGE
CHAPTER 3 (continued)	
DIR.....	3-40
DISKVERIFY.....	3-48
DOWNCOPY.....	3-53
FORMAT.....	3-56
INITSTATUS.....	3-63
JOBDELETE.....	3-65
LOCK.....	3-67
PERMIT.....	3-69
RENAME.....	3-74
RESTORE.....	3-77
SUBMIT.....	3-83
SUPER.....	3-87
TIME.....	3-90
UPCOPY.....	3-92
VERSION.....	3-95
CHAPTER 4	
HUMAN INTERFACE EXAMPLES	
Command Examples Format.....	4-1
How to Begin a Console Session.....	4-1
How to Create a Simple Data File.....	4-2
How to Copy Files.....	4-3
How to Copy to New Files.....	4-4
How to Display the Contents of Files.....	4-5
How to Replace Existing Files	4-5
How to Concatenate Files.....	4-6
How to Delete Files.....	4-8
How to Use Directories.....	4-9
How to Create a New Directory.....	4-9
How to Refer to a Directory.....	4-10
How to Add New Entries to a Directory.....	4-11
How to Create a Directory Within a Directory.....	4-12
How to List Directories.....	4-13
How to Move Files Between Directories.....	4-14
How to Delete a Directory.....	4-14
How to Change Your Default Directory.....	4-15
How to Rename Files and Directories.....	4-16
How to Rename Files.....	4-16
How to Rename Directories.....	4-18
How to Move Files Across Volume Boundaries.....	4-19
How to Format a New Volume.....	4-20
Diskette Switching Procedures.....	4-21
CHAPTER 5	
PATCHING UTILITY	
Types of Patches.....	5-1
Types of Replacement Code.....	5-1

CONTENTS (continued)

	PAGE
CHAPTER 5 (continued)	
Versions of the Patching Utility.....	5-2
Invoking the Patching Utility.....	5-2
Error Messages.....	5-4
Patching Procedures.....	5-5
Jump Instruction Patch.....	5-5
In-Place Patch.....	5-6
Patching Library Modules.....	5-7
Listing Translator Header Records.....	5-8
CHAPTER 6	
FILES UTILITY SYSTEM	
Hardware Required.....	6-1
Starting the Files Utility.....	6-2
Using the Files Utility.....	6-3
Changing Diskettes.....	6-3
Commands.....	6-3
ATTACHDEV (AD).....	6-3
BREAK (BR).....	6-4
CREATEDIR (CD).....	6-4
DELETE (DE).....	6-5
DETACH (DT).....	6-5
DIR (DI).....	6-5
DOWNCOPY (DC).....	6-6
FORMAT (FO).....	6-6
HELP (HE).....	6-9
UPCOPY (UC).....	6-9
Error Messages.....	6-9
APPENDIX A	
CONDITION CODES SUMMARY.....	A-1

FIGURES

2-1.	Example of a Named-File Tree.....	2-7
3-1.	Sample DEBUG Display.....	3-32
3-2.	FAST Directory Listing Example.....	3-43
3-3.	SHORT Directory Listing Example.....	3-43
3-4.	LONG Directory Listing Example.....	3-44
3-5.	EXTENDED Directory Listing Example.....	3-44
3-6.	INITSTATUS Display.....	3-63

TABLES

	PAGE
2-1. Input Pathname and Output Pathname Combinations.....	2-21
3-1. Human Interface Command Dictionary.....	3-5
3-2. Suggested Physical Device Names.....	3-9
3-3. Directory Listing Headings.....	3-45
A-1. iRMX [®] 86 Condition Codes.....	A-1

CHAPTER 1. LINE EDITING AND CONTROL CHARACTERS

Every terminal connected to an iRMX 86 application system communicates with the system via one of two software packages: the iRMX 86 Terminal Handler or the Terminal Support Code feature of the Basic I/O System.

The Terminal Handler is an independent layer of the Operating System that provides terminal I/O facilities for application systems that do not include the Basic I/O System. Because this manual assumes you are using an application system that includes the Basic I/O System, it does not discuss how to communicate with an application system via the Terminal Handler. Refer to the iRMX 86 TERMINAL HANDLER REFERENCE MANUAL for information about the line-editing and control characters available with the Terminal Handler.

The Terminal Support Code is a software package that interfaces to terminal device drivers to provide terminal communication for systems that include the Basic I/O System. This manual assumes that your terminal communicates with the iRMX 86 application system via the Terminal Support Code.

The Terminal Support Code provides a set of line-editing and control characters that give you the basic editing and control functions you need when entering text at a terminal. You can use these characters in addition to the Human Interface commands described later in this manual. This chapter discusses the line editing features and control characters that are available. However, the Terminal Support Code contains many features other than those discussed in this chapter. Refer to the iRMX 86 BASIC I/O SYSTEM REFERENCE MANUAL for a complete description of the Terminal Support Code.

TYPE-AHEAD

When you enter characters at the terminal, you can use the type-ahead feature to enter a number of lines at one time. The Terminal Support Code sends the first line to the Operating System for processing and stores additional lines in a type-ahead buffer. It sends the next line in the buffer to the Operating System after the Operating System finishes with the first line. If the type-ahead buffer becomes full, the Terminal Support Code sounds the terminal bell and refuses to accept input.

LINE EDITING AND CONTROL CHARACTERS

CONTROLLING INPUT TO A TERMINAL

The Terminal Support Code provides several characters that you can enter to control and edit terminal input. Some of these characters correspond to single keys on your terminal (such as carriage return or rubout). For others, called control characters, you must press the CTRL key, and while holding it down, also press an alphabetical key. This manual designates control characters as follows:

CTRL/character

The editing and control characters are processed by the Terminal Support Code. With the exception of the line terminator, they are not normally included in the input line that is sent to the Operating System.

The control characters listed in this section are the default characters. Each can be replaced with a different character by means of a selection procedure described in the iRMX 86 BASIC I/O SYSTEM REFERENCE MANUAL. The default editing and control characters for terminal input include:

CARRIAGE RETURN or LINE FEED	Terminates the current line and positions the cursor at the beginning of the next line. Entering either of these characters adds a carriage return/line feed pair to the input line.
RUBOUT	Deletes (or rubs out) the previous character in the input line. In response to the RUBOUT, your terminal display changes in one of two ways, depending on the configuration of the Terminal Support Code. In one configuration, each RUBOUT removes a character from the screen and moves the cursor back to that character position. In the other configuration, each RUBOUT echoes the deleted character back to the terminal. In the second configuration, also called <u>hard-copy mode</u> , the Terminal Support Code surrounds the echoed characters with the "#" character to distinguish the echoed characters from the surrounding text.
CTRL/p	A "quoting" character, which removes, from the character that follows it, any any meaning that is special to the Terminal Support Code. It literalizes the next character, causing it to be sent on to the Operating System, even if it is a control character that the Terminal Support Code understands. All control characters (except for output control characters) sent to the Operating System in this manner are not processed as control characters. Output control characters (such as CTRL/s and CTRL/q) perform their special functions even if preceded by a CTRL/p. The CTRL/p does not echo at the terminal.

LINE EDITING AND CONTROL CHARACTERS

CTRL/r	If the current input line is not empty, this character reprints the line with editing already performed. This enables you to see the effects of the editing characters entered since the most recent line terminator. If the current line is empty, this character reprints the previous line, up to the point of the line terminator. Additional CTRL/r characters display previous lines, until there are no more lines in the type-ahead buffer. Subsequent CTRL/r characters display the last line found.
CTRL/u	Discards the current line and the entire contents of the type-ahead buffer.
CTRL/x	Discards the current input line. This character echoes the "#" character, followed by a carriage return/line feed, at the terminal.
CTRL/z	If entered as the only character in a line, this character specifies an end-of-file, terminating a read from the terminal. If entered on a non-empty line, it terminates the line without appending a carriage return/line feed pair to the line.

CONTROLLING OUTPUT TO A TERMINAL

When sending output to a terminal, the Terminal Support Code always operates in one of four modes. You can switch the current output mode dynamically to any of the other output modes by entering output control characters. The output modes and their characteristics are as follows:

Normal	The Terminal Support Code accepts output from the application system and immediately passes the output to the terminal for display.
Stopped	The Terminal Support Code accepts output from the application system, but it queues the output rather than immediately passing it to the terminal.
Scrolling	The Terminal Support Code accepts output from the application system, and it queues the output as in the stopped mode. However, rather than completely preventing output from reaching the terminal, it sends a predetermined number of lines (called the <u>scrolling count</u>) to the terminal whenever the operator enters a control character at the terminal.
Discarding	The Terminal Support Code discards output from the application system without displaying or queuing the output.

LINE EDITING AND CONTROL CHARACTERS

The following control characters, when entered at the terminal, change the output mode for the terminal. Like the input control characters, these are defaults. They can be changed by a selection process described in the iRMX 86 BASIC I/O SYSTEM REFERENCE MANUAL.

- CTRL/o Places the terminal in discarding mode if the terminal is in a mode other than discarding mode. If the terminal is already in discarding mode, the CTRL/o character returns the terminal to its previous output mode.
- CTRL/q Resumes previous output mode. If you enter this character after stopping output with the CTRL/s character, output continues in the same manner as before you entered the CTRL/s (that is, if your terminal was in scrolling mode before you entered CTRL/s, output resumes in scrolling mode). Entering CTRL/q at any other time places your terminal in normal mode (that is, all output is displayed at the terminal without waiting for permission to continue). Therefore, you can use CTRL/q to reverse the effect of a CTRL/w and get your terminal out of scrolling mode.
- CTRL/s Places the terminal in stopped mode (stops output). You can resume output without loss of data by entering the CTRL/q character. If the terminal is in discarding mode (as a result of a CTRL/o character), the CTRL/s character has no effect on output.
- CTRL/t Places the terminal in scrolling mode and sets the scroll count to one. This means that you must enter another CTRL/t character after each displayed line in order to continue the display.
- CTRL/w Places the terminal in scrolling mode. In this mode, the terminal displays output several lines at a time (usually, enough lines to fill the screen) and then waits for user input to continue. When you enter another CTRL/w character, the terminal displays the next screen of information. The scrolling count is selectable; refer to the iRMX 86 BASIC I/O SYSTEM REFERENCE MANUAL for more information.

Entering the CTRL/w character while the terminal is already in scrolling mode increments the scrolling count by the original scrolling count value. Therefore, you can use CTRL/w to increase the number of lines the terminal displays before stopping. Entering an input line resets the scroll count to its original value.

An additional control character is supported which, although it doesn't affect the output mode of the terminal, can affect output to the terminal. This character is:

LINE EDITING AND CONTROL CHARACTERS

CTRL/c Deletes the type-ahead buffer and causes the Operating System to abort the currently-executing program. If you enter a Human Interface command to initiate a program, you can enter CTRL/c to stop it.

ESCAPE SEQUENCES

The Terminal Support Code also accepts escape characters that you can enter to further define your terminal. (For example, you could set the scroll count or switch your terminal into transparent mode so that control characters have no effect.) You can enter these escape characters from the terminal, or you can write them to the terminal from a program. Refer to the iRMX 86 BASIC I/O SYSTEM REFERENCE MANUAL for more information about these escape characters.

CHAPTER 2. USING THE HUMAN INTERFACE

This chapter discusses how to use the Human Interface. It doesn't provide detailed descriptions of individual commands; that is done in Chapter 3. However, it does address the following topics:

- Requirements for including the Human Interface in your system.
- Configurable features of the Human Interface.
- The process of accessing the Human Interface.
- The iRMX 86 file structure and file-naming conventions (including wild cards).
- The general syntax of a command.
- The system manager.

REQUIREMENTS

The Human Interface is a layer of the iRMX 86 Operating System. To include the human Interface in your application system, you must also include the following additional layers:

- Nucleus
- Basic I/O System
- Extended I/O System
- Application Loader

During command execution, the Human Interface invokes the services of these other iRMX 86 layers in a way that is transparent to the operator. Therefore, an operator needs little or no knowledge of operating system structures to load and execute programs from the console keyboard. For more information about iRMX 86 configuration, refer to the iRMX 86 CONFIGURATION GUIDE.

CONFIGURABLE FEATURES OF THE HUMAN INTERFACE

The Human Interface, like the other layers of the iRMX 86 Operating System, is configurable. Thus, any description of how to use the Human Interface depends a great deal on its configuration. This manual describes several features of the Human Interface that may be different (or not present at all) in your system. The configurable items that are the most visible to the operator include:

USING THE HUMAN INTERFACE

- Multi-access. If your Human Interface is configured for multi-access, several users can access the Human Interface at once via separate terminals. One of the users, the system manager, has more capabilities than other users and is responsible for managing system resources and controlling who can use the system. Users of a multi-access Human Interface are concerned about user IDs, access rights to files, and attaching and detaching devices -- all in relation to the other users of the system.

However, if your Human Interface is configured for single access, you are less interested in much of this information. You are the only user accessing the system; therefore you are not as interested in user IDs and the system manager. You have no great concern about file access rights since all the files on the system are yours.

This manual attempts to satisfy both users. It explains all the information that the user of a multi-access Human Interface needs, but it also points out cases where information does not apply to users of single-access systems. In all cases, the information required by a user of a single-access Human Interface is a subset of the information required by a user of a multi-access system.

- Initial program. During initialization, the Human Interface starts an initial program for each terminal. This initial program is a Command Line Interpreter (CLI), a program that reads commands and starts those programs running.

This manual assumes that the initial program for all users is the CLI supplied by the Human Interface. If your Human Interface is configured with a different initial program, the information in this manual might not describe your Human Interface accurately. The system prompts might be different, the command syntax might be different, or you might be restricted to using a special program such as an interpreter or a transaction processor. If you suspect that your initial program is not the standard CLI, contact the person who configured your system to determine the differences.

There are other configuration options that affect how the system appears to a user. When describing these items, this manual points out their configurable nature and urges you to consult the iRMX 86 CONFIGURATION GUIDE. If you are not involved in iRMX 86 configuration, contact the person who configured your system to obtain more information.

LOADING THE OPERATING SYSTEM

Before you can access the Human Interface, someone must first load the Operating System into the memory of your iRMX 86 system and start it running. This process can vary from system to system, but generally it involves one of the following procedures:

USING THE HUMAN INTERFACE

- Connecting the target system (the iRMX 86 system) to an Intel Microcomputer Development System and using the iSBC 957B package to load the Operating System from development system files to memory in the iRMX 86 system. This procedure is normally done during the development phase of an application system, when some of the system elements remain unstable. Refer to the USER'S GUIDE FOR THE iSBC 957B iAPX 86, 88 INTERFACE AND EXECUTION PACKAGE for more information.
- Using the iRMX 86 Bootstrap Loader to load the Operating System from iRMX 86 files to memory in the iRMX 86 system. For some systems, you simply ensure that an iRMX 86-formatted volume containing the Operating System resides in the proper device and reset the system. On other systems, you must verify that the device contains the correct volume and perform the following steps:

1. Reset the system; usually, this involves pressing the RESET button on the system chassis. A series of characters (usually asterisks) should appear at the system terminal (the one connected to the processor board).
2. Type an uppercase U at the system terminal. This accesses the iAPX 86, 88 monitor. The monitor displays the following information:

```
iAPX 86, 88 MONITOR Vx.y
```

```
.
```

The period (.) is the monitor prompt.

3. Use the monitor's B command to bootstrap load the Operating System. In most cases you do this by entering:

```
.B
```

For the default configuration of the Bootstrap Loader, this command loads a file with pathname SYSTEM/RMX86 from the first available device. If your Operating System resides on a file with a different pathname, you must specify that pathname in the B command. Refer to the iRMX 86 LOADER REFERENCE MANUAL, the iRMX 86 CONFIGURATION GUIDE, and the USER'S GUIDE FOR THE iSBC 957B iAPX 86, 88 INTERFACE AND EXECUTION PACKAGE for more information about the configuration of the Bootstrap Loader and the interface between it and the iSBC 957B package.

Some systems contain the entire Operating System in ROM and do not require you to load additional information from secondary storage. The usual process for starting these systems is simply to reset the system.

If you were not involved in the configuration of your system and are unsure about how to load and start the Operating System, contact the person who configured your system.

ACCESSING THE HUMAN INTERFACE

Assuming that the Operating System software is loaded into the system, you access the Human Interface by powering on your terminal. If your application system is configured for automatic baud rate recognition, you must also enter the following character at the keyboard:

U (uppercase U)

This character allows the Operating System to determine the baud rate of your terminal.

When the Human Interface starts running, it creates an environment for you to enter commands. This environment is an iRMX 86 job, which this manual refers to as an interactive job.

As part of creating this interactive job, the Human Interface assigns you a user ID. This user ID is your "identity" in the system. It determines your access to files and devices. Whenever you create files, the Human Interface assigns your user ID as the owner ID of the file. Being the owner of a file gives you complete control over the file; you can read it, delete it, write it, update it, and select the access that you wish to grant to other users. Your own ability to access files created by other users depends on the access they grant you.

At this point, an initial program begins execution. The initial program that runs in your interactive job (at your terminal) may be different from one that runs at another terminal. (A configuration option specifies which initial programs are associated with which user IDs.) Initial programs are command line interpreters (CLIs), which read and parse command input and start programs running based on that input. The Human Interface supplies a standard CLI, which this manual assumes you are using. The standard CLI begins running by displaying the following (configurable) header message and prompt:

```
iRMX 86 HI CLI, Vx.x: user = <user ID>
```

-

where:

Vx.x	The version number of the Human Interface.
user = <user ID>	A display of your user ID. The Human Interface uses this ID to determine the type of access you have to files and devices. Most single-access systems are set up to give you an ID of WORLD (65535 decimal), but some may differ. The user

USING THE HUMAN INTERFACE

ID WORLD is compatible with multi-access systems (if transferring files is necessary), because every multi-access user has read and write access to files created by WORLD.

- (hyphen) The Human Interface prompt. This prompt implies that the CLI is ready to accept command input.

If the information that appears at your terminal is different from this, contact the person who configured your iRMX 86 Operating System to determine the differences between your initial program and the standard CLI.

Next, the standard CLI searches for the logon file, a file whose pathname is :PROG:R?LOGON (later sections of this chapter discuss pathnames of files). There can be a file with this name for each user of the system. The CLI expects to find command invocation lines in this file. When it finds this file, the CLI automatically invokes the SUBMIT command to process all the commands in the file (refer to Chapter 3 for more information about SUBMIT). You can modify the information in your :PROG:R?LOGON file to change the amount of processing that occurs automatically when the Operating System recognizes your terminal.

After processing all the commands in the logon file, the CLI issues its prompt (-) and returns control to you. At this point you can enter Human Interface commands and invoke programs.

FILE STRUCTURE

One of the primary uses of Human Interface commands is manipulating files. Before you can use the Human Interface commands described in Chapter 3, you should have an understanding of the kinds of files that exist in an iRMX 86 environment and how to access those files.

TYPES OF FILES

There are three basic types of files in an iRMX 86 environment: named files, physical files, and stream files. These files are used as follows:

- | | |
|-------------|---|
| Named files | Named files divide the data on mass storage devices into individually-accessible units. Users and programs refer to these files by name when they want to access information stored in them. Terminal operators access named files more often than any other file type. |
|-------------|---|

USING THE HUMAN INTERFACE

Physical files Physical files are mechanisms by which the Operating System can access an entire I/O device as a single file. The Human Interface accesses backup volumes and devices such as line printers and terminals in this manner. It also accesses secondary storage devices (such as disk drives) as physical devices when formatting them. When terminal operators access physical files, it is usually in a manner that is transparent to them (such as copying a named file to the line printer or formatting a disk).

Stream files Stream files are mechanisms for communicating between programs. Two programs can use a stream file for communication if one program writes information to the stream file while another program reads the information. Terminal operators seldom use stream files directly.

When manipulating data with Human Interface commands, you are most often dealing with named files. Therefore it is important that you know about the hierarchy of named files and file-naming conventions. The next sections discuss these topics in detail.

NAMED FILE HIERARCHY

The iRMX 86 Operating System allows you to organize named files into structures called file trees, as shown in Figure 2-1.

As you can see from the figure, there are two kinds of files in the file tree: data files and directories. Data files, (shown as triangles in Figure 2-1) contain the information that you manipulate in the course of your terminal session (for example, inventory, accounts payable, text, source code, and object code). Directories, (shown as rectangles in Figure 2-1) contain only pointers to other files (either named files or directories). The iRMX 86 Operating System allows you to have multiple directories in a hierarchical structure so that instead of having a single directory containing an enormous number of files, you can organize your files into logical groupings under several directories. You can display the list of files in any directory by invoking the DIR command for that directory (refer to Chapter 3 for more information).

Another advantage of hierarchical file structure is that duplicate file names are permitted unless the files reside in the same directory. Notice in Figure 2-1 that the file tree contains two directories named BILL. (These directories are on the extreme left and extreme right of the figure.) However, the Operating System recognizes them as unique files because each resides in a different directory.

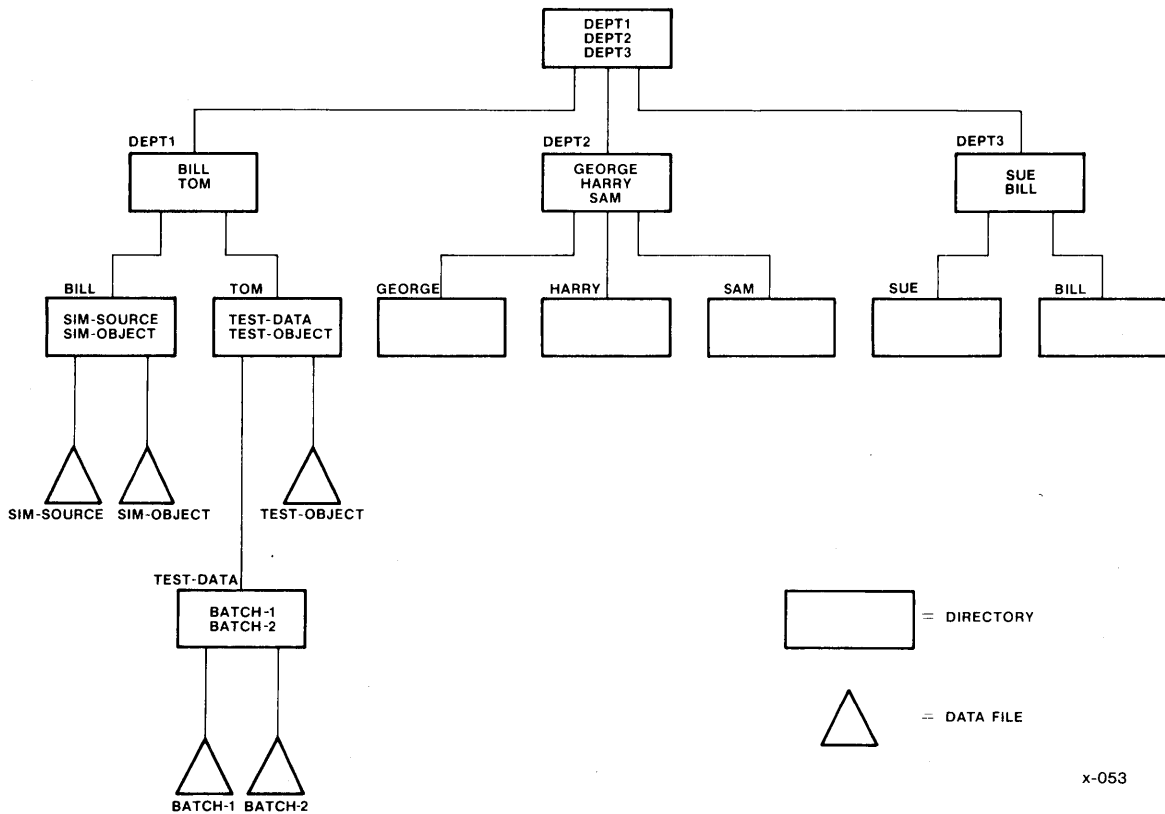


Figure 2-1. Example of a Named-File Tree

Each file tree resides on a secondary storage volume -- the storage medium that contains the data. Examples of volumes include flexible diskettes, hard disks, and bubble memories. Before you can place named files on a volume, you must format the volume to accept named files. The formatting process writes a number of data structures on the volume to aid the Operating System in creating and maintaining files. You can use the `FORMAT` command (described in Chapter 3) to format your volumes.

The uppermost point of each file tree is a directory called the root directory. When formatted for named files, each secondary storage volume has one and only one root directory. For this reason:

- There can be only one file tree per secondary storage volume.
- A file tree cannot extend to more than one volume.

USING THE HUMAN INTERFACE

PATHNAMES

This section describes how to specify a particular file in a named-file tree. For simplification, it assumes that all files reside in the same file tree, and thus in the same volume. To identify the volume as well as the file, you must include a logical name for the device as the first portion of the file specification. Refer to the "Logical Names" section, later in this chapter, for more information about logical names.

In a file tree, each file (data or directory) has a unique shortest path connecting it to the root directory. For example, in Figure 2-1, the shortest path from the root directory to file BATCH-2 goes through directory DEPT1, through directory TOM, through directory TEST-DATA, and finally stops at data file BATCH-2. When you want to perform an operation on a file (for example, using the COPY command to copy one file to another), you must specify not only the file's name, but the path through the file tree to the file. This is called the file's pathname. For file BATCH-2 in Figure 2-1, the pathname is:

```
DEPT1/TOM/TEST-DATA/BATCH-2
```

This pathname consists of the names of files (in uppercase or lowercase characters; the Operating System treats them as the same) and separators. In this case, slashes (/) separate the individual components of the pathname and tell the Operating System that the next component resides down one level in the file tree. You can use another separator, the circumflex or up-arrow (^), between path components. Each circumflex tells the Operating System that the next path component resides up one level in the file tree. The following pathname, although not the shortest possible pathname, indicates another path to file BATCH-2:

```
DEPT1/BILL^TOM/TEST-DATA/BATCH-2
```

If you always start at the root directory, the circumflex separator is not very useful, since you usually want to traverse down the file tree. However, in some systems, your starting point in the file tree may be a directory other than the root directory. In such cases the circumflex separator is useful in accessing files in other branches of the file tree. Your default prefix (discussed later in the "Logical Names" section of this chapter) determines your starting point in the file tree.

For example, suppose your starting point in the file tree is the directory TOM shown in Figure 2-1. In order for you to access a file in directory BILL from this starting point, you must use the circumflex in the pathname. To indicate file SIM-SOURCE in directory BILL, you could enter the pathname:

```
^BILL/SIM-SOURCE
```

This path tells the Operating System to go up one level in the file tree from the starting point (to directory DEPT1 from directory TOM), search in that directory for directory BILL, and search in directory BILL for file SIM-SOURCE.

Another way to specify files in different branches of the file tree is by including the slash separator as the first character in the pathname. This tells the Operating System to ignore your normal starting point and begin the path from the root directory. Using the previous example where the starting point is directory TOM, another way to specify SIM-SOURCE is with the pathname:

```
/DEPT1/BILL/SIM-SOURCE
```

The initial slash causes the Operating System to search in the root directory for directory DEPT1 instead of in the normal starting directory (TOM).

LOGICAL NAMES

Although the Operating System allows you to use pathnames to refer to files, it also allows you to create symbolic names that correspond to files or devices. These symbolic names are called logical names. You can create logical names that represent devices, data files, or directories. After creating a logical name, you can refer to the entity it represents by specifying the logical name. The rules for logical names are:

- Each logical name must contain 1 to 12 ASCII characters.
- The hexadecimal representation of each character must be between 021h and 07Fh inclusive (printing characters).
- The logical name cannot include the characters colon (:), slash (/), up-arrow or circumflex (^), asterisk (*), and question mark (?).
- When you specify a logical name, you must surround it with colons.

When referring to logical names, this manual always lists the surrounding colons.

For an example of how to use logical names, refer again to Figure 2-1. Suppose you have created a logical name called :ME: that represents the pathname DEPT1/TOM/TEST-DATA (a later paragraph in this section discusses how to create this logical name). If you want to refer to the directory TEST-DATA, you can either specify its pathname as before, or you can specify the logical name :ME:. If you want to refer to the file BATCH-1 under directory TEST-DATA, you can do this in either of the following ways:

```
DEPT1/TOM/TEST-DATA/BATCH-1
```

or

```
:ME:BATCH-1
```

USING THE HUMAN INTERFACE

The second line shows that you can use a logical name as a beginning portion (or prefix) of a pathname. The logical name tells the Operating System where to begin in its search for the file. However, you cannot use a logical name in the middle or at the end of a pathname. If you use a logical name, you must specify it at the beginning.

Notice that you must not include a slash or circumflex between the logical name and the next path component if you want the Operating System to search down one level. If you include the slash, the Operating System ignores the normal starting point (the directory TEST-DATA) and searches for the file BATCH-1 in the root directory of the volume. If you include the circumflex, the Operating System searches up one level from the starting point.

As a Human Interface user, you deal with two general classes of logical names: logical names for devices and logical names for files.

Logical Names for Devices

Device logical names allow you to refer to specific devices by name. The Operating System can establish logical names for devices during system initialization. You can establish other logical names for new or existing devices by invoking the ATTACHDEVICE command (see Chapter 3 for details).

By using device logical names as the prefix portion of your pathname specifications, you can refer to any file on any device. For example, suppose your system contains two flexible disk drives for which you have established logical names :F0: and :F1:. (You used the ATTACHDEVICE command to attach the devices as :F0: and :F1:.) If you have a diskette containing the file DEPT2/HARRY, you could place the diskette in drive :F0: and access the file with the pathname:

```
:F0:DEPT2/HARRY
```

If you put the same diskette in drive :F1:, you could access the file by specifying the pathname:

```
:F1:DEPT2/HARRY
```

You can see that for devices containing named files, the device logical name is actually a logical name for the root directory on that device. If you enter the DIR command (described in Chapter 3) to list the directory of device :F1:, as follows:

```
DIR :F1:
```

DIR actually displays the root directory of the volume in drive :F1:.

Logical Names for Files

A logical name for a file provides a shorthand way of accessing that file. For example, suppose you have a file that resides several levels down in the file tree, such as:

```
:F1:DEPT1/TOM/TEST-DATA/BATCH-2
```

where :F1: is logical name for the device that contains the file. After entering this pathname a few times, you might find it inconvenient to continually enter so many characters. If so, you can establish a logical name for this pathname, such as :BATCH:. (You could also say that you attached the file with the logical name :BATCH:.) Then, whenever you want to refer to the file in a command, you can specify the logical name instead of the pathname.

If your logical names refer to directories instead of data files, you can use the logical names in the prefix portion of a pathname. For example, consider the same pathname:

```
:F1:DEPT1/TOM/TEST-DATA/BATCH-2
```

Suppose you have attached the pathname :F1:DEPT1/TOM/TEST-DATA as logical name :TEST:; therefore it is a logical name for the directory TEST-DATA. To refer to file BATCH-2, you could enter:

```
:TEST:BATCH-2
```

Logical names for files come into existence in two ways. One way is for you to invoke the ATTACHFILE command (refer to Chapter 3 for details). The other way is for the Operating System to create them. The Operating System establishes a number of logical names for files during system initialization. A later section lists these logical names.

Where Logical Names are Stored

When the Operating System creates logical names, at initialization time or as a result of ATTACHFILE or ATTACHDEVICE commands, it does so by placing the logical name, along with a token for a connection to the file or device, into an object directory. This process is referred to as cataloging the logical name (refer to the iRMX 86 EXTENDED I/O SYSTEM REFERENCE MANUAL for more information about this process). The object directory that receives this information determines the scope of the logical name (that is, who can use the logical name). There are three possibilities:

Root object directory	Logical Names cataloged in the object directory of the root job can be accessed by every user. When you use ATTACHDEVICE to create logical names for devices, the Operating System catalogs the logical names in the root directory.
-----------------------	--

Logical names cataloged in the root object directory remain valid until deleted or until the system is reinitialized.

Global object directory Logical names can be cataloged in the object directory of a job that is designated as a global job (refer to the iRMX 86 EXTENDED I/O SYSTEM REFERENCE MANUAL for more information about global jobs). Each interactive job (user session) is a global job. When you use ATTACHFILE to create logical names for files, the Operating System catalogs the logical names in your global job. Likewise, if you invoke any commands that issue ATTACHFILE commands (such as a SUBMIT command), the Operating System catalogs the logical names in your global job. You (and any commands that you invoke) can use the logical names cataloged in your interactive job. However, other users have no access to these logical names.

Logical names cataloged in your interactive job remain valid for the life of your interactive job or until deleted.

Local object directory Logical names can be cataloged in the object directory of the job itself. When you invoke a command (such as DIR), the Operating System creates a job for that command and catalogs certain objects in its object directory. A command that you create and invoke might also use iRMX 86 system calls to catalog logical names in its own object directory.

Logical names cataloged in a local job remain valid only for the life of the job or until deleted.

Whenever you (or one of the commands you invoke) use a logical name, the Operating System searches for that logical name in as many as three different object directories. It first looks in the local object directory. If the logical name is not defined there, it next looks in the global object directory and finally, if necessary, the root object directory. It uses the first such logical name it finds.

Because of this order of search, you can override the system logical names (those cataloged in the root object directory) by cataloging the same logical names (but representing different files or devices) in the object directory of your interactive job. For example, suppose you used the ATTACHFILE command to attach a file with the logical name :SYSTEM:. Then, whenever you specify :SYSTEM:, the Operating System refers to your file and not the one represented by the same logical name in the root object directory.

USING THE HUMAN INTERFACE

Logical Names Created by the Operating System

The Operating System establishes several logical names that you can use without first having to create them. It catalogs some of these logical names in the root object directory (where they are available to all users). It catalogs others in global object directories (these are specific to each interactive job). It catalogs others in local object directories (these are specific to each interactive job and to each command invoked).

The Human Interface catalogs system-wide logical names in the root object directory. These logical names are available to all users and they represent the same file or device for all users. The number of logical names created and their identities depend on the configuration of your Operating System. However, the following logical names are available on most systems.

- :BB:** A device that is treated as an infinite sink (byte bucket). Anything written to :BB: disappears, and anything read from :BB: returns an end-of-file.
- :LANG:** A directory used to store language products, such as assemblers, compilers, and linkers.
- :SD:** The system device. If you used the Bootstrap Loader to load your system, this logical name refers to the device from which the Bootstrap Loader read the Operating System file.
- :STREAM:** The prototype stream file connection. To create a connection to a stream file, you must use this logical name as the prefix portion of the pathname.
- :SYSTEM:** The directory containing the Human Interface commands.
- :UTILS:** A directory used to store utility programs created by users.
- :WORK:** A directory that Intel language translators and utilities use to store their temporary and work files.

The following logical names are cataloged in each user's global object directory. Although each user has access to these names, the names represent different files or devices for each user.

- :\$:** Your default prefix. This is the path to your default directory. If you do not specify a logical name (a prefix) at the beginning of a pathname, the Operating System automatically uses \$: as the prefix. In this case, the Operating System assumes that the file resides in the directory corresponding to \$:. During an interactive session, you can use the ATTACHFILE command to change the directory corresponding to \$:.

USING THE HUMAN INTERFACE

:HOME: Your default prefix when you first start using the Human Interface. Initially, **:HOME:** and **:\$:** represent the same directory. This logical name provides you with the ability to re-establish your original **:\$:** logical name if you become lost in the hierarchical file structure. You should not use **ATTACHFILE** to change the directory corresponding to **:HOME:.**

:PROG: A directory in which to store your programs.

The following logical names are cataloged in the local object directory of each user and each command that a user invokes. These logical names can have different meanings for each user and each command.

:CI: The terminal keyboard (or command input). As the name implies, each user's **:CI:** refers to the terminal associated with that user.

:CO: The terminal screen (or command output). As the name implies, each user's **:CO:** refers to the terminal associated with that user.

Upon initialization, your Human Interface may create additional logical names. These logical names are configuration parameters. Contact the person who configured your system for more information about the logical names initially available to you. The **iRMX 86 CONFIGURATION GUIDE** discusses this subject in more detail.

Removing Volumes from Devices

Removing volumes from devices (such as removing flexible diskettes from drives) destroys any connections that may have existed to files on that device. Therefore, any logical names that represent files on the volume are no longer valid once you remove the volume. The names remain cataloged in the directories, but they do not represent valid connections. Therefore, before removing volumes, you should invoke **DETACHFILE** commands to detach the files.

WILD CARDS

Wild cards provide a shorthand notation for specifying several files in a single reference when entering commands. You can use either of two special wild card characters in the last component of a pathname to replace some or all characters in that component. The wild card characters are:

USING THE HUMAN INTERFACE

? The question mark matches any single character. The Human Interface allows any character to appear in that character position. It selects every file that meets this requirement. For example, the name "FILE?" could imply all of the following files:

```
FILE1
FILE2
FILEA
```

* The asterisk matches any number of characters (including zero characters). The Human Interface allows any number of characters to appear in that character position. It selects every file that meets this requirement. For example, the name "FILE*" could imply all of the following files:

```
FILE1
FILE.OBJ
FILE
FILECHANGE
```

You can use multiple wild cards in a single pathname. For example, the name:

```
?PIF?.*
```

matches every file whose second through fourth characters are "PIF" and whose sixth character is a period. These files could include all of the following names (or more):

```
RPIFC.LIB
EPIFL.TXT
HPIFC.
```

You can use wild cards in both input pathnames (files that commands read for information) and output pathnames (files into which commands write information). For example, in the command:

```
COPY A* TO B*
```

the A* represents the input pathname and B* represents the output pathname. In this command (which copies information from one file to another), the Human Interface searches the appropriate directory for all files that begin with the "A" character. Then it copies each file to a file of the same name, but beginning with the "B" character. If the directory contains the files:

```
ALPHA
A112
A
```

the previous command would copy files in the following manner:

```
ALPHA TO BLPHA
A112 TO B112
A TO B
```

USING THE HUMAN INTERFACE

There are several operational characteristics that you should be aware of when using wild cards:

- Wild cards are valid in the last component of the pathname only. Therefore, :F1:SYSTEM/APP1/FILE* is a valid pathname, but :F1:SYSTEM/APP*/FILE1 is not valid.
- You can negate the meaning of a wild card character by enclosing it in quotes, either single (') or double ("). For example, if you have a file named F*123, you can refer to it alone in a command by specifying F*'123 or 'F*123'.
- When you specify input and output pathnames in commands, you can specify lists of pathnames, separated by commas. For example:

```
COPY A,B,C TO D,E,F
```

copies A to D, B to E, and C to F. If you use a wild cards in any one of the output pathnames, you must use the same wild cards in the same order in the corresponding input pathname. The term "same order" means that if you use both the "*" and the "?" characters, their ordering must be the same in both the input and output pathnames. For example, the following is valid:

```
COPY A*B?C* TO *DE?FGH*I
```

However, the following is invalid because the wild cards are out of order:

```
COPY A*B?C* TO *DE*FGH?I
```

- If you use wild cards in an input pathname, you can omit all wild cards from the corresponding output pathname to cause the Human Interface to perform file concatenation. For example, suppose a directory contains files A1, B1, and C1. The following command is valid:

```
COPY *1 TO X
```

It copies files in the following manner:

```
A1 TO X  
B1 AFTER X  
C1 AFTER X
```

Refer to the "Command Syntax" section later in this chapter for more information about the prepositions TO and AFTER.

- The "*" character matches as close to the end of the pathname as possible. For example, suppose the directory contains the file "ABXCDEFXGH", and you enter the command:

```
COPY *X* TO *1*
```


USING THE HUMAN INTERFACE

This command copies:

```
ABXCDEFXGH TO ABXCDEF1GH
```

The first asterisk matches the characters "ABXCDEF", and the second asterisk matches the characters "GH".

COMMAND SYNTAX

This section describes the general syntax rules that apply when entering Human Interface commands at a terminal. These rules apply equally to both the supplied Human Interface commands and any user-created commands that may have been added to your system. The individual command descriptions in Chapter 3 contain additional and more specific information about each supplied Human Interface command.

The elements that form a standard command entry include a command name, required input parameters (if any), and optional parameters. The general structure of a command line is as follows (brackets [] indicate optional portions):

```
command-name [inpath-list [preposition outpath-list]] [parameters] cr
```

where:

command-name	Pathname of the file containing the command's executable object code.
inpath-list	One or more pathnames, separated by commas, of files to be read as input during command execution.
preposition	A word that tells the executing command how to handle the output. The four prepositions used in Intel-supplied commands are TO, OVER, AFTER, and AS.
outpath-list	One or more pathnames, separated by commas, of files that are to receive the output during command execution.
parameters	Parameters that cause the command to perform additional or extended services during command execution.
cr	A line terminator character. This character terminates the current line and causes the cursor to go to a new line. The RETURN (or CARRIAGE RETURN) key and NEW LINE (or LINE FEED) key are both line terminators.

You can enter all elements of a command line in uppercase characters, lowercase characters, or a mix of both. The Human Interface makes no distinction between cases when it reads command line items. In addition, you can include the following optional command line entries:

USING THE HUMAN INTERFACE

continuation mark An ampersand character (&) indicates that the command continues on the next line. When you include the ampersand character, the Human Interface displays two asterisks (**) on the next line to prompt for the continuation line. All characters appearing after the continuation mark but before the line terminator are interpreted as comments.

Within available memory limits, you can use as many continuation lines for a given command as you desire. After you enter the line terminator without a preceding ampersand character, the invoked command receives the entire command string as a single command.

comment character A semicolon (;) character indicates that all text following it on the current line is a non-executable comment. You can also enter comments after a continuation mark (&) but before the line terminator. A common use of comments in commands is in a SUBMIT file list of commands (see the SUBMIT command in Chapter 3).

quoting characters Two single-quote (') or double-quote (") characters remove the semantics of special characters they surround. For example, if you surround an ampersand character (&) with single quotes, the ampersand is not recognized as a continuation character. The same holds for other characters such as asterisk (*), question mark (?), equals (=), semicolon (;), and others. The only special characters not affected by the quoting characters are the pathname separators (/ and ^), semicolon (:), and dollar sign (\$). Although you can use either single quotes or double quotes as quoting characters, you must use the same quoting character at the beginning and at the end of your quoted string. If you want to include the quoting character inside your quoted string, you must specify the character twice. For example:

```
'can't'
```

You can accomplish the same effect by using the other quoting character, for example:

```
"can't"
```

Although the Human Interface places no restriction on the number of characters in a command, each terminal line can have a maximum of 255 characters, including any punctuation, embedded blanks, continuation mark, non-executable comments, and carriage return. If your command requires more characters, use continuation lines.

The following sections discuss the individual elements of the command syntax in more detail.

USING THE HUMAN INTERFACE

COMMAND NAME

Each Human Interface command is a file of executable code that resides in secondary storage. When you specify a command name, you actually specify the name of the file containing the command's code. If you write your own command (refer to the iRMX 86 HUMAN INTERFACE REFERENCE MANUAL for information), you invoke it by entering the name of the file that contains it. After you invoke a command, the Operating System loads it from secondary storage into memory and executes it in conformance with parameters you specify.

When you enter a command name, you can enter the complete pathname of the command, or, in many cases, you can enter just the last component of the pathname.

- If you enter the complete pathname of the command (that is, if you include a logical name as the prefix portion of the pathname), the Operating System searches only the device and directory you specify for the command. If it cannot find the command there, it returns an error message.
- If you enter only the last component of the pathname (such as COPY instead of :F1:SYSTEM/COPY), the Operating System automatically searches a certain number of directories for the command. It does not return an error message until it has searched each of the directories. The number of directories searched and the order of search are Human Interface configuration parameters. However, in the default case, the Operating System searches the following directories, in order, for commands:

```
:$:  
:PROG:  
:SYSTEM:
```

When writing your own commands, you can take advantage of the order in which the Operating System searches directories. For example, suppose you write your own copy command, one that provides more or different functions than the Human Interface COPY command. If you want to invoke your own program whenever you type the command "COPY", you can simply place your copy program in a file called COPY in your :PROG: directory. Since the Operating System searches the :PROG: directory before searching the :SYSTEM: directory (the directory that normally contains Human Interface commands), it will invoke your copy program when you enter the command "COPY".

If you still want to be able to invoke the Human Interface COPY command, you can do so by entering its complete pathname, that is, by entering the following:

```
:SYSTEM:COPY
```

USING THE HUMAN INTERFACE

PREPOSITIONS

Preposition parameters in a command line tell the the command how you want it to process the output file or files. The Human Interface commands usually provide three options in the choice of a preposition: TO, OVER, and AFTER. The preposition AS is also available for use in the ATTACHDEVICE and ATTACHFILE commands. The TO preposition and :CO: (console screen) will be used by default if you do not specify a preposition and an output file. The prepositions have the following meaning:

TO Causes the command to send the processed output to new files; that is, to files that do not already exist in the given directory. If a listed output file already exists, the command displays the following query at the console screen:

<pathname>, already exists, OVERWRITE?

Enter a Y or y if you wish to write over the existing file. Enter any other character if you do not wish the file to be overwritten. In the latter case, the command does not process the corresponding input file but rather goes to the next input file in the command line. Commands process input files and write to output files on a one-for-one basis. For example:

```
COPY A,B TO C,D
```

copies file A to file C and file B to file D.

OVER Causes the command to write your input files to the output files in sequence, destroying any information currently contained in the output files. It creates new output files if they do not exist already. For example:

```
COPY SAMP1,SAMP2 OVER OUT1,OUT2
```

copies the data from file SAMP1 over the present contents of file OUT1, and copies the data of SAMP2 over the contents of file OUT2.

AFTER Causes the command to append the contents of one or more files to the end of one or more new or existing files (file concatenation). For example:

```
COPY IN1,IN2 AFTER DEST1,DEST2
```

appends the contents of file IN1 to the the end of file DEST1, and appends the contents of IN2 to the end of DEST2.

AS A special preposition used with the ATTACHDEVICE and ATTACHFILE commands. When you use the AS preposition, the Operating System does not assume that the command contains input pathnames and output pathnames. Rather, it sees the parameters as entities that it must associate (for example, ATTACHFILE associates a pathname with a logical name).

USING THE HUMAN INTERFACE

INPATH-LIST AND OUTPATH-LIST

An inpath-list specifies the files on which a command is to operate. An outpath-list defines the destination or destinations of the processed output. Each inpath-list or outpath-list consists of a pathname (or logical name) or list of pathnames. If you specify multiple pathnames, you must separate the individual pathnames with commas. Embedded blanks between pathnames are optional. You can also use wild cards to indicate multiple pathnames (refer to the "Wild Cards" section of this chapter).

Usually when you specify multiple pathnames, each pathname in the inpath-list has a corresponding pathname in the outpath-list. For example, the command:

```
COPY A, B TO C, D
```

copies file A to file C and also copies file B to file D. Therefore, A and C are corresponding pathnames, and so are B and D. However, there are some instances when the number of input pathnames you enter differs from the number of output pathnames. The validity of the operation depends on whether the pathname lists contain single pathnames, lists of pathnames, a wild-card pathname, or lists of wild-card pathnames. Table 2-1 lists the possibilities and describes the Human Interface's action in each instance. The following sections discuss the Human Interface's actions in more detail.

Table 2-1. Input Pathname and Output Pathname Combinations

Inpath-list	Outpath-list	Human Interface Action
single pathname	single pathname	one-for-one match
single pathname	list of pathnames	error
single pathname	wild-card pathname	error
single pathname	list of wild cards	error
list of pathnames	single pathname	concatenate
list of pathnames	list of pathnames	one-for-one match
list of pathnames	wild-card pathname	error
list of pathnames	list of wild cards	error
wild-card pathname	single pathname	concatenate
wild-card pathname	list of pathnames	error
wild-card pathname	wild-card pathname	one-for-one match
wild-card pathname	list of wild cards	error
list of wild cards	single pathname	concatenate
list of wild cards	list of pathnames	concatenate
list of wild cards	wild-card pathname	concatenate
list of wild cards	list of wild cards	one-for-one match

USING THE HUMAN INTERFACE

One-For-One Match

The combinations in Table 2-1 that are marked "one-for-one match" are those in which each element in the inpath-list is matched with an element of the outpath-list. An example of this is the command:

```
COPY A*, B* TO C*, D*
```

In this case, the Human Interface copies all files beginning with the character "A" to corresponding files beginning with the character "C". When it finishes this operation, it advances past the comma to the next set of pathnames (copies all files beginning with "B" to corresponding files beginning with "D").

Concatenate

The combinations in Table 2-1 that are marked "concatenate" are those in which there are multiple input pathnames that correspond to a single output pathname. In this situation, the Operating System automatically appends the remaining input files to the end of the specified output file, regardless of the preposition you specify.

This allows you to combine one-for-one file operations (as in TO or OVER preposition) with file concatenation (as in the AFTER preposition) in a single command, and thus avoid entering an extra command to perform a separate concatenation operation. The following example explains this situation.

Assume that in a COPY command, you use the TO preposition and specify the following input and output pathnames:

```
COPY A,B,C TO D
```

When the Human Interface processes the command line, it copies file "A" to file "D" and appends files "B" and "C" to the end of file "D" as follows:

```
A TO D
B AFTER D
C AFTER D
```

Notice that this concatenation occurs only when there are multiple elements in the inpath-list that correspond to a single element of the outpath-list. This means that the following commands are invalid:

```
COPY A, B, C TO D, E ; INVALID COMMAND
COPY A*, B*, C* TO D*, E* ; INVALID COMMAND
```

Error Conditions

The combinations in Table 2-1 that are marked "error" indicate invalid operations. For these combinations, the Human Interface returns an error message without performing the requested operation.

OTHER PARAMETERS

Most commands allow you to enter parameters other than inpath-lists, outpath-lists, and prepositions. These other parameters are known as keyword parameters, because you must enter a particular word, called a keyword, to obtain the additional or extended services provided by the parameter.

For example, the DIR command (described in Chapter 3) lists the contents of a directory. You can enter several different keyword parameters to specify the amount of information displayed and the format of the display. A command such as:

```
DIR :SYSTEM: EXTENDED
```

displays the contents of the :SYSTEM: directory in extended format. You could substitute other keywords such as SHORT or LONG to obtain different formats.

The command descriptions in Chapter 3 list the keyword parameters available with each command. However, the descriptions list the complete names for the keywords. When you use keywords, you can enter their complete names or you can enter only as many characters as are necessary to uniquely identify the keyword. For example, you could enter the previous command as:

```
DIR :SYSTEM: E
```

For the DIR command, the character E uniquely identifies the EXTENDED parameter. Other keywords might require additional characters to make them unique.

Some keyword parameters also require an associated value. An example of this is the FORMAT command (described in Chapter 3), which prepares secondary storage volumes for iRMX 86 use. A command such as:

```
FORMAT :F1:TEST FILES = 60
```

formats a volume on device :F1: and sets up the volume to contain at most 60 files. The keyword in this command (FILES) has an associated value (60). Although this example and the descriptions in Chapter 3 use the equal sign (=) to associate keywords and values, there are actually two ways to do this. They are:

USING THE HUMAN INTERFACE

keyword = value
keyword (value)

The blanks are optional. You can use either method when entering Human Interface commands.

SYSTEM MANAGER

The multi-access Human Interface supports a user called the system manager. The system manager's primary purpose is to maintain the multi-access configuration files. The system manager can modify these files to add or delete user IDs, add or delete terminals, and change terminal or user characteristics (refer to the iRMX 86 CONFIGURATION GUIDE for more information). For security reasons, no user other than the system manager can access these files.

In addition, the system manager has a special user ID which gives that user privileges that other users do not have. The system manager:

- Has read access to all data files and list access to all directories.
- Can change the access rights of any file, regardless of the file's owner.
- Can detach devices attached by any user.
- Can delete any user from the system.

Any operator can become the system manager by invoking a Human Interface command called SUPER. This command (which requires entering a password) changes the operator's user ID from its normal value to that of the system manager. Once an operator invokes SUPER, that operator has all the powers of the system manager. Refer to Chapter 3 for more information about the SUPER command.

CHAPTER 3. HUMAN INTERFACE COMMANDS

The commands described in this chapter are supplied by Intel for iRMX 86 Operating Systems that are configured with the Human Interface. If you are a new user of the Human Interface, it is suggested that you review the information on file-naming conventions and invocation considerations in Chapter 2 before reading this chapter.

This chapter does not describe how to specify the names of the devices and directories that contain the Human Interface commands. This is because during the Human Interface configuration process you can specify a number of directories that the Human Interface automatically searches for commands. If you place your Human Interface commands in one of these directories (normally the :SYSTEM: directory), you can invoke the commands by entering only their names. However, if your commands reside in a directory that the Human Interface does not search automatically, or if you have multiple commands with the same name in different directories, you can use the complete pathname for the command. For example, if the DIR command resides in directory COMMANDS on device :F6: (a directory not normally searched by the Human Interface), you can invoke the command by entering:

```
:F6:COMMANDS/DIR
```

Refer to the iRMX 86 CONFIGURATION GUIDE for more information about Human Interface Configuration.

This chapter presents the commands in alphabetical sequence without regard for functional organization. The Human Interface Command Dictionary (Table 3-1) also lists a functional grouping of the commands for fast reference.

ERROR MESSAGES

Each command can generate a number of error messages which indicate errors in the way you specified the command. The messages that apply to a specific command are listed with that command. However, the following are general error messages that can appear with many of the commands:

- command not found

There is no file whose pathname is the same as the command name you specified, nor can the Human Interface find the file in any of the directories it automatically searches.

- <logical name>, device does not belong to you

The device you specified was originally attached by a user other than WORLD or you.

HUMAN INTERFACE COMMANDS

- <pathname>, file does not exist
The pathname you specified does not represent an existing file.
- <pathname>, invalid file type
You specified a data file for an operation that required a directory, or vice versa.
- <logical name>, invalid logical name
The logical name you specified contains unmatched colons, is longer than 12 characters, or contains invalid characters.
- <pathname>, invalid pathname
The pathname you specified contains invalid characters or a component of the pathname (other than the last one) does not exist or does not represent a directory.
- <logical name>, is not a device connection
The logical name you specified does not represent a connection to a physical device.
- <logical name>, logical name does not exist
The logical name you specified does not exist.
- parameters required
The command you specified cannot be entered without parameters.
- program version incompatible with system
The command and the Operating System are not compatible. The command expects to obtain information from internal tables that are not present. Therefore the command cannot run successfully.
- <control>, unrecognized control
The parameter you entered is not valid for the specified command.

HUMAN INTERFACE COMMANDS

- `<exception value> : <exception mnemonic>`, while loading command

The Operating System encountered an exceptional condition while attempting to load the command into memory from secondary storage. The message lists the exception code encountered.

- `<exception value> : <exception mnemonic>`

An operational error occurred during the execution of the command. The `<exception value>` and `<exception mnemonic>` portions of the message indicate the exception code encountered.

- `<parameter>, <exception value> : <exception mnemonic>`

The command encountered an exceptional condition while attempting to process the `<parameter>` portion of the command. The `<exception value>` and `<exception mnemonic>` portions of the message indicate the exception code encountered.

COMMAND SYNTAX SCHEMATICS

The syntax for each command described in this chapter is presented by means of a "railroad track" schematic, with syntactic elements scattered along the track. Your entrance to any given schematic is always from left to right, beginning with some command name entry.

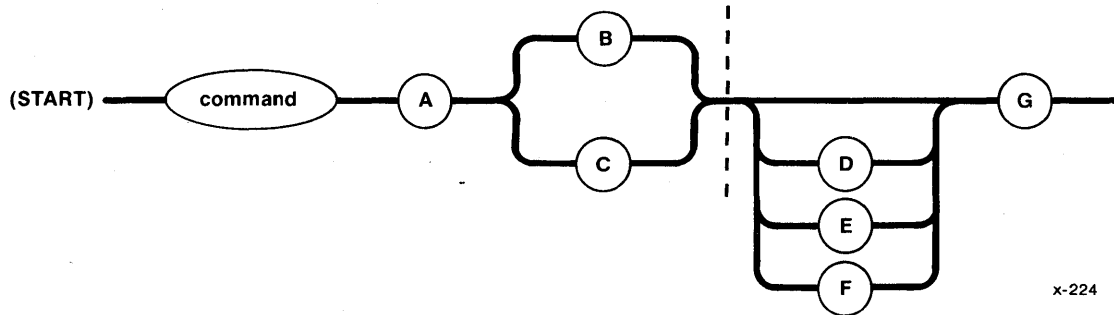
Elements shown in uppercase characters must be typed in a command line exactly as shown in the command schematics except that you can type them either in uppercase or lowercase characters; the Human Interface makes no distinction between cases in alphabetic characters. Syntactic elements shown in lowercase characters are generic terms, which means that you supply the specific item, such as the pathname for a file.

The vertical dotted line separates the position-dependent parameters from those that are position-independent. Parameters to the left of the dotted line must be entered in the order listed (from left to right). Parameters to the right of the dotted line can be entered in any order (as long as they obey the rest of the syntax).

The example that follows shows all the possible paths through a railroad track schematic. Notice that the main track goes through required elements in a given command.

"Railroad sidings" go through optional parameter elements. In some cases, you have a choice of going through one of several possible sidings before returning to the main track. In still other cases, the main track itself diverges into two separate tracks, which means that you must select one parameter or the other but not both.

HUMAN INTERFACE COMMANDS



In this example:

- A is a required element. It is position-dependent; it must be entered first.
- Either B or C is required but not both. These elements are also position-dependent. Whichever element you enter must follow A immediately.
- D, E, or F are all optional but only one can be selected. These are position-independent elements. If you select one of these elements, you can enter it before or after G.
- G is required. It is a position-independent parameter. You can enter it before or after D, E, or F.

HUMAN INTERFACE COMMANDS

Table 3-1. Human Interface Command Dictionary

Command	Synopsis	Page
File Management Commands		
ATTACHFILE	Associates a logical name with an existing file.	3-13
COPY	Creates new data files, or copies files to other pathnames.	3-24
CREATEDIR	Creates one or more new directories.	3-28
DELETE	Deletes data files and empty directories from a volume on secondary storage.	3-33
DETACHFILE	Removes the association of a logical name with a file.	3-38
DIR	Lists a directory's filenames (and optionally, file attributes).	3-40
DOWNCOPY	Copies files and directories from an iRMX 86 volume mounted on a secondary storage device to an ISIS-II secondary storage device.	3-53
PERMIT	Grants or rescinds user access to a file.	3-69
RENAME	Renames files or directories.	3-74
UPCOPY	Copies files and directories from an ISIS-II secondary storage device to an iRMX 86 volume mounted on a secondary storage device.	3-92
Volume Management Commands		
ATTACHDEVICE	Attaches a new physical device to the system and catalogs its logical name to the root job's object directory.	3-7
BACKUP	Copies named files to a backup volume.	3-16
DETACHDEVICE	Removes a physical device from system use and deletes its logical name from the root job's object directory.	3-35
DISKVERIFY	Verifies the data structures of named and physical volumes.	3-48

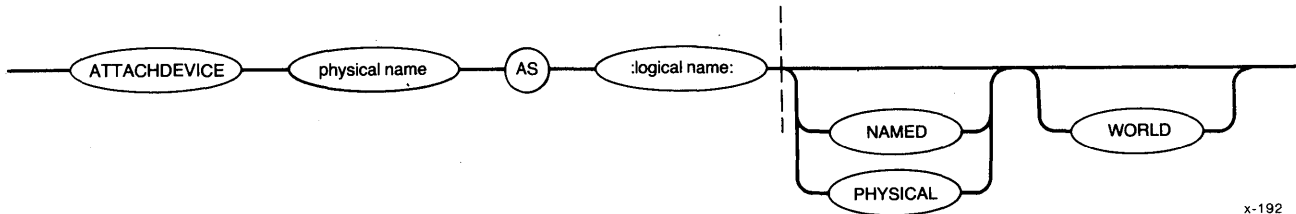
HUMAN INTERFACE COMMANDS

Table 3-1. Human Interface Command Dictionary (continued)

Command	Synopsis	Page
Volume Management Commands (continued)		
FORMAT	Formats an iRMX 86 volume.	3-56
RESTORE	Copies files from a backup volume to a named volume.	3-77
Multi-Access Commands		
INITSTATUS	Displays the initialization status of Human Interface terminals.	3-63
JOBDELETE	Deletes a running interactive job.	3-65
LOCK	Prevents the Human Interface from automatically creating an interactive job after the job has been deleted.	3-67
SUPER	Changes the operator's user ID into that of the system manager (user ID 0) and grants the ability to change to other user IDs.	3-87
General Utility Commands		
DATE	Sets or resets the system date, or displays the current date.	3-29
DEBUG	Transfers control to the iSBC 957B package to debug an iRMX 86 application program.	3-31
SUBMIT	Reads, loads, and executes a string of commands from secondary storage instead of the keyboard.	3-83
TIME	Sets or resets the system clock, or displays the current system time.	3-90
VERSION	Displays the version numbers of commands.	3-95

ATTACHDEVICE

This command attaches a physical device to the Operating System and associates a logical name with the device. The command catalogs the logical name in the root object directory, making the logical name accessible to all users. The format of the command is as follows:



x-192

INPUT PARAMETERS

- physical name** Physical device name of the device to be attached to the system. This name must be the name used in one of the Basic I/O System's Device Unit Information Blocks (DUIB), as defined at system configuration time (see Table 3-2).
- AS** Preposition; required for the command.
- :logical name:** A 1- to 12-character name, that represents the logical name to be associated with the device. Colons surrounding the logical name are optional; however, if you use colons, you must use matching colons.
- NAMED** Specifies that the volume mounted on the device is already formatted for NAMED files. Examples of volumes that can contain named files are diskettes or hard disk platters. If neither NAMED nor PHYSICAL are specified, NAMED is the default. See the FORMAT command in this chapter for a further description of NAMED files.
- PHYSICAL** Specifies that the volume mounted on the logical device is considered to be a single, large file. Examples include line printers and terminals. See the FORMAT command in this chapter for a further description of PHYSICAL volumes.

WORLD Specifies that user ID WORLD (65535 decimal) is the owner of the device. This implies that any user can detach the device. If you omit this parameter, your user ID is listed as the owner of the device. In this case, only you and the system manager can detach the device.

DESCRIPTION

ATTACHDEVICE attaches a device to the system and catalogs a logical name for it in the root job's object directory. The logical name is the means by which all users can access the device. Devices must have their characteristics listed in the Basic I/O System's Device Unit Information Block (DUIB) at configuration time before they can be attached with the ATTACHDEVICE command.

Table 3-2 lists the physical device names normally used with the Basic I/O System. Your system might support a subset of these devices or it might support devices not listed. If it supports the devices listed, it might support them under different names. Therefore, consult the person who configured your system to determine the correct device names for your system.

One frequent use of the ATTACHDEVICE command is to attach a new device, such as a new disk drive or a line printer, without having to reconfigure portions of the Operating System. (See the DETACHDEVICE command in this chapter for a description of how to detach a device from the system without reconfiguring.)

Unless you have a user ID of WORLD (65535) or specify the WORLD parameter, once you attach a device, only you and the system manager can detach the device. This prevents users from detaching devices belonging to other users and prevents you from accidentally detaching system volumes. However, if you have a user ID of WORLD or specify the WORLD parameter, any device that you attach can be detached by any other user. Refer to the DETACHDEVICE command for more information.

When the device attachment is completed, the ATTACHDEVICE command displays the following message:

<physical name>, attached as <logical name>, id = <user id>

where <physical name> and <logical name> are as specified in the ATTACHDEVICE command and <user id> is your user ID (or WORLD, if you specify the WORLD parameter).

Table 3-2. Suggested Physical Device Names

Physical Device Names	Controller	Device Type	Unit Number	Sides	Density	Bytes per Sector
Flexible Disk Drives						
FO	204	Shugart SA800	0	1	Single	128
F1	204	Shugart SA800	1	1	Single	128
FX0	204	Shugart SA800	0	1	Single	512
FX1	204	Shugart SA800	1	1	Single	512
AFO	208	Shugart SA800	0	1	Single	128
AF1	208	Shugart SA800	1	1	Single	128
AFDO	208	Shugart SA800	0	1	Double	256
AFD1	208	Shugart SA800	1	1	Double	256
AMFO	208	Shugart SA410	0	1	Double	256
AMF1	208	Shugart SA410	1	1	Double	256
AFDD0	208	Shugart SA850/SA851	0	2	Double	256
AFDD1	208	Shugart SA850/SA851	1	2	Double	256
AFDX0	208	Shugart SA850/SA851	0	2	Double	1024
AFDX1	208	Shugart SA850/SA851	1	2	Double	1024
WFO	218	Shugart SA800	0	1	Single	128
WF1	218	Shugart SA800	1	1	Single	128
WFDO	218	Shugart SA800	0	1	Double	256
WFD1	218	Shugart SA800	1	1	Double	256
WMFO	218	Shugart SA410	0	1	Double	256
WMF1	218	Shugart SA410	1	1	Double	256
WFDD0	218	Shugart SA850/SA851	0	2	Double	256
WFDD1	218	Shugart SA850/SA851	1	2	Double	256
WFDX0	218	Shugart SA850/SA851	0	2	Double	1024
WFDX1	218	Shugart SA850/SA851	1	2	Double	1024
Hard Disk Drives						
DO	206		0			512
D1	206		1			512
DS0	206		0			128
DS1	206		1			128
Winchester Disk Drives						
IWO	215	Priam 3450				1024
MWO	215	Memorex 101				1024
PWO	215	Pertec D8000				1024
SWO	215	Shugart SA1002				1024

Table 3-2. Suggested Physical Device Names (continued)

Physical Device Names	Controller	Device Type	Unit Number	Sides	Density	Bytes per Sector
Storage Module Disk Drives						
SMD0	220		0			1024
SMD1	220		1			1024
Bubble Memory Device						
B0	254	4 bubbles				256
Others						
BB		Byte bucket (already attached)				
STREAM		Stream file device (already attached)				
T0	USART	terminal				
T1-T4	534	terminals				
C0	270	terminal				

ERROR MESSAGES

- <device name>, cannot be ATTACHED as <type> device

The device specified by <device name> cannot support the type of files specified by <type> (NAMED or PHYSICAL). ATTACHDEVICE does not attach the device. For example, the NAMED option is not valid for a device such as a line printer.

- <device name>, device already attached

The specified device has already been attached. ATTACHDEVICE does not attach the device.

- <device name>, device does not exist

The physical device name you specified does not correspond to a name the Basic I/O System recognizes. That is, the person who configured your application system did not specify <device name> as the name of a device-unit during configuration of the Basic I/O System. ATTACHDEVICE does not attach the device.

- <logical name>, logical name already exists

The specified logical name is already cataloged in the root job's object directory. ATTACHDEVICE does not attach the device.

- 0085 : E\$LIST, too many device names

You tried to attach more than one physical device with a single ATTACHDEVICE command. ATTACHDEVICE does not attach a device.

- <logical name>, volume is not a NAMED volume

ATTACHDEVICE attempted to attach a device as a named device and discovered a physical volume on the device. However, ATTACHDEVICE does attach the device. You can use the device after formatting the volume as a named volume or after inserting a named volume in the device.

- <logical name>, volume not formatted
<logical name>, <exception value> : <exception mnemonic>

ATTACHDEVICE attempted to attach a device as a named device and encountered an I/O error while searching for the volume's root directory. This usually indicates that the volume is not formatted. However, ATTACHDEVICE does attach the device.

- <logical name>, volume not mounted

The specified device does not contain a volume. However, ATTACHDEVICE does attach the device.

- <exception value> : <exception mnemonic>, while collecting device name

ATTACHDEVICE encountered an exceptional condition while parsing the device name from the command line. This message lists the resulting exception code. ATTACHDEVICE does not attach the device.

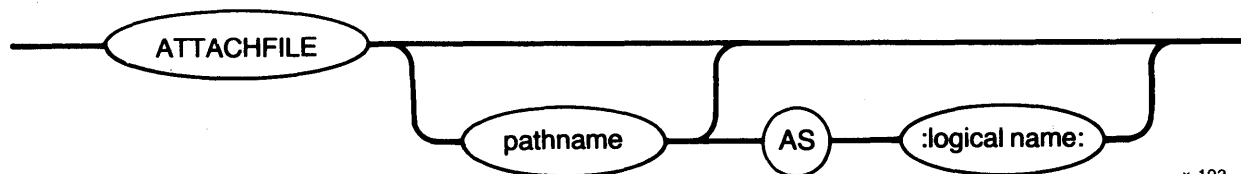
ATTACHDEVICE

- <exception value> : <exception mnemonic>, while collecting logical name

ATTACHDEVICE encountered an exceptional condition while parsing the logical name from the command line. This message lists the resulting exception code.

ATTACHFILE

This command allows you to associate a logical name with an existing file. The command catalogs the logical name in your global object directory. The format of this command is as follows:



x-193

INPUT PARAMETERS

pathname	Pathname of the file to which the Human Interface associates a logical name.
:logical name:	1- to 12-character name that represents the logical name to be associated with the file. Colons surrounding the logical name are optional; however, if you use colons, you must use matching colons. If you omit this parameter, the default logical name is :\$:.

If you enter the ATTACHFILE command without parameters, the default is:

```
ATTACHFILE :HOME: AS :$:
```

DESCRIPTION

The ATTACHFILE command allows you to associate a logical name with an existing file. After making this association, you can use the logical name, instead of the entire pathname, to refer to the file.

When the attachment is complete, ATTACHFILE displays the following message:

```
<pathname>, attached AS <logical name>
```

where <pathname> and <logical name> are as specified in the ATTACHFILE command.

ATTACHFILE makes the association between a file and a logical name by cataloging a connection to the file in your global object directory (this is normally the object directory of your interactive job). It catalogs the connection under the name specified as the logical name. If there is another connection cataloged in the object directory under the same logical name, ATTACHFILE uncatalogs and deletes the previous connection before cataloging the new one. If an object other than a

ATTACHFILE

connection is cataloged in the directory under the specified logical name, ATTACHFILE leaves the previous object as is, does not catalog the new connection, and displays an error message to describe the situation.

Because ATTACHFILE catalogs the connection in your global object directory, the logical name has effect only within your interactive job. Therefore, several users can specify the same logical name without affecting the others.

If you specify a pathname for a file but omit the logical name, ATTACHFILE attaches the file as `:$:`. This allows you to change your default prefix. Changing your default prefix can be useful when you want to manipulate files that reside in a directory other than the one specified by your original default prefix. For example, suppose you have a file that you normally refer to as:

```
:PROG:SOURCE/PLM/INTERRUPT/TEST.P86
```

You can change your default prefix with the command:

```
ATTACHFILE SOURCE/PLM/INTERRUPT
```

Then, you can refer to the file as simply:

```
TEST.P86
```

When you finish using the files in directory `:PROG:SOURCE/PLM/INTERRUPT`, you can return your default prefix to its original setting by entering:

```
ATTACHFILE
```

This is the same as entering:

```
ATTACHFILE :HOME: AS :$:
```

`:HOME:` is a logical name that refers to the same directory as your original default prefix. Therefore, you can change your default prefix as much as you like with ATTACHFILE and return to the original setting by making reference to `:HOME:.` However, you cannot use ATTACHFILE to change the meaning of `:HOME:.` (Also, you cannot use ATTACHFILE to change the meaning of `:CI:` and `:CO:.`)

The logical name created with ATTACHFILE remains valid until one of the following situations occur:

- A DETACHFILE command (described later in this chapter) dissolves the association between file and logical name.
- The interactive session that specified the ATTACHFILE command terminates processing. This occurs when a user, in response to the Human Interface prompt, enters a Control-Z character to reinitialize the interactive job. In this case, the Operating System deletes the interactive job and then recreates it.

- A task deletes the connection to the file via a Basic I/O System or Extended I/O System call (refer to the iRMX 86 BASIC I/O SYSTEM REFERENCE MANUAL or the iRMX 86 EXTENDED I/O SYSTEM REFERENCE MANUAL for more information about connections). In this instance, the logical name remains cataloged in the global directory, but the connection to which it refers does not exist.
- A user forcibly detaches the volume containing the file via the DETACHDEVICE command (described later in this chapter).
- A user removes the volume from the drive.

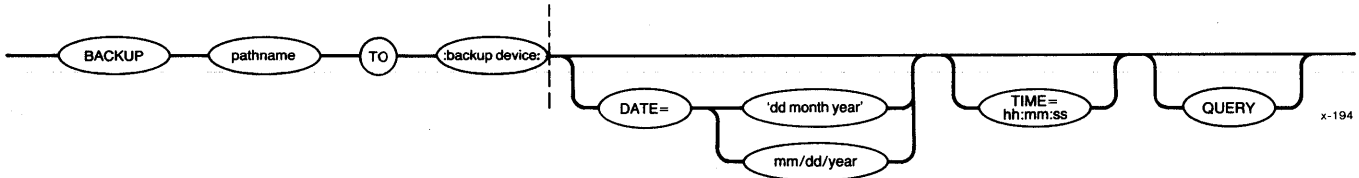
ERROR MESSAGES

- <logical name>, list of logical names not allowed
You entered more than one logical name as input to ATTACHFILE.
- <pathname>, list of pathnames not allowed
You entered more than one pathname as input to ATTACHFILE.
- <logical name>, logical name not allowed
You attempted to attach a file using a logical name :HOME:, :CI:, or :CO:. You cannot change the meaning of these logical names.
- <logical name>, not a file connection
The logical name you specified, <logical name>, is already cataloged in object directory of the session and does not represent a connection object.
- <pathname>, not allowed as default prefix
You attempted to attach a physical or stream file as your default prefix (:\$:). Only named files are valid.
- <logical name>, too many logical names
Your global object directory is full. Therefore ATTACHFILE is unable to catalog the file's name in the object directory.

BACKUP

This command saves files from a named volume by copying them to a physical volume which serves as a backup volume. Later, you can use the RESTORE command (described later in this chapter) to retrieve these files and copy them to named volumes.

The format of this command is as follows:

**INPUT PARAMETERS**

pathname

Pathname of a file on the source volume. BACKUP saves files from the branch of the file tree that begins with the specified file. If you specify the logical name of the device only, BACKUP saves all files in the volume, beginning with the root directory.

'dd month year'

One form of the date parameter that BACKUP uses, in conjunction with the time parameter, to determine which files to save. BACKUP saves only those files that have been modified since the specified date and time. If you use this form of the date parameter, you must enclose the date parameter in single quotes. The individual fields of this parameter are:

dd Two-digit number that specifies the day of the month.

month Designation for the month. You can enter the whole name (such as AUGUST) or enough characters to distinguish one month from another (for example, AU, to distinguish AUGUST from APRIL). You can use this form for specifying the month only when using the "dd month year" format.

year Designation for the year. You can enter this as a two- or four-digit number, as follows:

<u>entered year</u>	<u>actual year</u>
0 through 77	2000 through 2077
78 through 99	1978 through 1999
100 through 1977	error
1978 through 2099	1978 through 2099
2100 and up	error

If you omit the date parameter but specify the time parameter, the date defaults to the current system date. If you omit both the date and time parameters, the date defaults to 1 JAN 78.

mm/dd/year

Alternate form of the date parameter. If you use this form, you do not have to surround the parameter with quotes. The individual fields of this parameter are:

mm Numerical designation for the month (for example: 1 represents January, 2 represents February, etc.). You can use this form for specifying the month only when using the "mm/dd/year" format.

dd Same as in the previous form of the date parameter.

year Same as in the previous form of the date parameter.

hh:mm:ss

Time parameter that BACKUP uses, in conjunction with the date parameter, to determine which files to save. BACKUP saves only those files that have been modified since the specified date and time. The individual fields of this parameter are:

hh Hours specified as 0-24.

mm Minutes specified as 0-59.

ss Seconds specified as 0-59.

If you omit this parameter, the time defaults to 00:00:00.

QUERY

Causes the Human Interface to prompt for permission to save each file. The Human Interface prompts with one of the following queries:

<pathname>, BACKUP data file?

or

<pathname>, BACKUP directory?

Enter one of the following responses to the query:

<u>Entry</u>	<u>Action</u>
Y or y	Save the file.
E or e	Exit from the BACKUP command.
R or r	Continue saving files without further query.
Any other character	If data file, do not save the file; if directory file, do not save the directory or any file in that portion of the directory tree. Query for the next file, if any.

OUTPUT PARAMETER

:backup device: Logical name of the device to which BACKUP copies the files.

DESCRIPTION

BACKUP is a utility which saves named files on backup volumes, such as diskettes. BACKUP saves the following information for each file:

- File name
- Access list, including owner
- Extension data
- File granularity
- Contents of the file

You can copy this information back to a named file by using the RESTORE utility, described later in this chapter.

Before a volume can be used as a backup volume, the volume must be formatted. Although BACKUP will accept both physical and named volumes, it is recommended that you use freshly-formatted physical volumes or old backup volumes for this purpose. BACKUP issues a message before continuing if the backup volume you supply is anything other than a freshly-formatted physical volume. When BACKUP copies files to the backup volume, it overwrites any information that currently exists on the volume.

In order for BACKUP to save files from a named volume, you must have read access to the files and to the directories that contain them.

You can limit the files which BACKUP processes in the following ways:

- If you specify a complete directory name instead of just the device's logical name in the invocation line, BACKUP limits its processing to the specified directory and its subdirectories.
- If you specify the date and time parameters, BACKUP processes only those files modified since the specified time.
- If you specify the QUERY parameter, BACKUP asks permission before saving each file. If you deny permission for BACKUP to save a data file, BACKUP skips the file and continues with the next file. If you deny permission for BACKUP to save a directory file, BACKUP skips the directory and all files contained in the directory or its subdirectories.

When you enter the BACKUP command, BACKUP displays the following sign-on message:

```
IRMX 86 DISK BACKUP UTILITY, Vx.y
```

where Vx.y is the version number of the utility. It then displays the following message:

```
all files modified after <date>, <time> will be saved
```

where <date> and <time> are the values you specified in the date and time parameters (or the defaults). Then BACKUP prompts you for a backup volume.

Whenever BACKUP requires a new backup volume, it displays the following message:

```
<backup device>, mount backup volume #<nn>, enter Y to continue:
```

where <backup device> indicates the logical name of the backup device and <nn> the number of the requested volume. (BACKUP in some cases displays additional information to indicate problems with the current volume.) In response to this message, place a volume in the backup device and enter one of the following:

BACKUP

<u>Entry</u>	<u>Action</u>
Y, y, R or r	Continue the backup process.
E or e	Exit from the BACKUP command.
Any other character	Invalid entry; reprompt for entry.

BACKUP continues prompting for a backup volume until you supply one that it can access.

If the backup volume you supply is not a freshly-formatted physical volume, but one that BACKUP can access (such as a named volume, a previously-used backup volume, or a physical volume containing data), BACKUP informs you of this with one of the following messages:

<backup device>, not a physical volume, enter Y to overwrite:

or

<backup device>, backup volume #<nn>, <date>, <time>, enter Y to overwrite:

or

<backup device>, named volume, <volume name>, enter Y to continue:

where <backup device> is the logical name of the backup device, <volume name> is the volume name of the named volume, <nn> is the volume number of the backup volume, and <date> and <time> are the date and time on which the previous backup was performed. In response to these messages, enter one of the following:

<u>Entry</u>	<u>Action</u>
Y, y, R, or r	Use the volume as a backup volume, overwriting the information currently stored on the volume.
E or e	Exit from the BACKUP command.
Any other character	Reprompt for another volume.

As BACKUP saves each file in the source volume, it displays one of the following message at your console output device (:CO:):

<pathname>, saved

or

<pathname>, directory saved

When the backup process is complete, BACKUP displays the number of data files saved, as follows:

files saved = <num>

If your backup volume becomes full and you supply additional backup volumes, you should write the numbers of the backup volumes on the volume labels. Later, when you restore files to a named volume with the RESTORE utility, you must supply the backup volumes in order.

ERROR MESSAGES

- <backup device>, backup operation not completed

When BACKUP requested a new backup volume, you specified an "E" to exit BACKUP. This message is a reminder that the backup operation is not complete. The last file on the last backup volume may be incomplete.

- <backup device>, backup volume #<nn>, <date>, <time>, enter Y to overwrite:

The backup volume you supplied already contains backup information. BACKUP lists the logical name of the backup device, the volume number, and the date on which the original backup occurred. It overwrites this volume if you enter Y, y, R, or r.

- <backup device>, cannot attach volume
<backup device>, <exception value> : <exception mnemonic>

<backup device>, mount backup volume #<nn>, enter Y to continue:

BACKUP cannot access the backup volume. This could be because there is no volume in the backup device or because of a hardware problem with the device. The second line of the message indicates the iRMX 86 exception code encountered. BACKUP continues to issue this message until you supply a volume that BACKUP can access.

- <pathname>, <exception value> : <exception mnemonic>, cannot back up file

For some reason BACKUP could not copy a file from the named volume, possibly because you do not have read access to the file or because there is a faulty area on the named volume. The message lists the pathname of the file and the exception code encountered. BACKUP copies as much of the file as possible and continues with the next file.

- <backup device>, device in use
<backup device>, <exception value> : <exception mnemonic>

The device you specified for the backup device is the same device that contains your input pathname. Continuing would result in damage to the files on the input volume.

- <backup device>, error writing volume label
<backup device>, <exception value> : <exception mnemonic>
- <backup device>, mount backup volume #<nn>, enter Y to continue:

When BACKUP attempted to write a label on the backup volume, it encountered an error condition, possibly because of a faulty area on the volume, or because the volume is write-protected. The second line of the message indicates the iRMX 86 exception code encountered. BACKUP reprompts for a different backup volume.

- <backup device>, input and output are on same device

The device you specified for the backup device is the same device that contains your input pathname. Continuing would result in damage to the files on the input volume.

- <backup device>, invalid backup device

The logical name you specified for the backup device was not a logical name for a device. Examples of invalid names are :CI:, :CO:, and :HOME:.

- <exception value> : <exception mnemonic>, invalid DATE or TIME

For either the DATE or TIME parameter, you entered a value that is out of range (such as 31 FEB 81 or 26:03:62). The message lists the exception code encountered as a result of this entry.

- <backup device>, named volume, <volume name>, enter Y to overwrite:

The backup volume you supplied is a named volume. BACKUP lists the logical name of the device containing the volume and the volume name. It overwrites this volume if you enter Y, y, R, or r.

- <backup device>, not a physical volume, enter Y to overwrite:

The backup volume you supplied is a formatted volume, but it has a label that is not readable. BACKUP will overwrite this volume if you enter Y, y, R, or r.

- output specification missing

You did not supply the logical name of the backup device when you entered the BACKUP command.

- <exception value> : <exception mnemonic>, requested date/time later than system date/time

The date and time you specified is more recent than the current system date and time (as set by the DATE and TIME commands). Either the date and time you specified in the BACKUP command are in error or you did not set the system date and time.

- <pathname>, too many input pathnames

You attempted to enter a list of pathnames or use a wild-carded pathname as the input pathname. You can enter only one pathname per invocation of BACKUP.

- <pathname>, too many output pathnames

You attempted to enter a list of logical names for the backup device. You can enter only one output logical name per invocation of BACKUP.

- <pathname>, unable to complete directory

BACKUP encountered an error when accessing a file in the <pathname> directory. It skips the rest of the files in the directory and goes on to the next directory. This error could occur if you do not have list access to the directory.

- <backup device>, volume not formatted

<backup device>, mount backup volume #<nn>, enter Y to continue:

The backup volume you supplied was not formatted. BACKUP continues to issue this message until you supply a formatted backup volume.

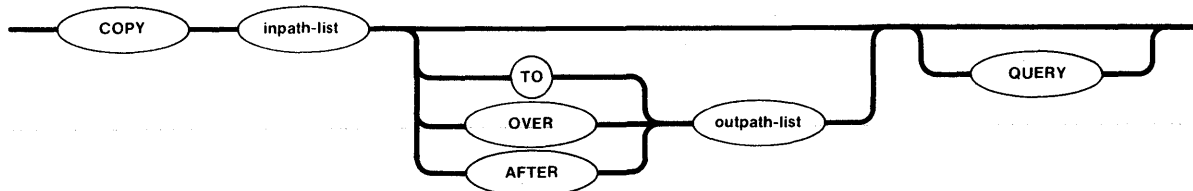
- <backup device>, write error on backup volume
<backup device>, <exception value> : <exception mnemonic>

BACKUP encountered an error condition when writing information to the backup volume. The second line of the message lists the exception code encountered. This error is probably the result of a faulty area on the volume.

COPY

This command reads data from the specified input source or sources and writes the output to the specified destination file or files.

The format of the command is as follows:

**INPUT PARAMETERS****inpath-list**

One or more pathnames for the files to be copied. Multiple pathnames must be separated by commas. Separating blanks are optional. To copy files on a one-for-one basis, you must specify the same number of files in the inpath-list as in the outpath-list.

QUERY

Causes the Human Interface to prompt for permission to copy each file. Depending on the specified preposition (TO, OVER, or AFTER), the Human Interface prompts with one of the following queries:

<pathname>, copy TO <out-pathname>?

<pathname>, copy OVER <out-pathname>?

<pathname>, copy AFTER <out-pathname>?

Enter one of the following (followed by a carriage return) in response to the query:

<u>Entry</u>	<u>Action</u>
Y or y	Copy the file.
E or e	Exit from COPY command
R or r	Continue copying files without further query.
Any other character	Do not copy this file; go to the next file in the input list.

OUTPUT PARAMETERS

TO	<p>Writes the listed input files to named new output files. The specified output file or files should not already exist. If they do, COPY displays the following message:</p> <p style="padding-left: 40px;"><pathname>, already exists, OVERWRITE?</p> <p>Enter Y, y, R, or r if you wish to write over the existing file. Enter an "N" (upper or lower case) or a carriage return alone if you do not wish to overwrite the existing file. In the latter case, the COPY command will pass over the corresponding input file without copying it, and will attempt to copy the next input file to its corresponding output file.</p> <p>If you specify multiple input files and a single output file, COPY appends the remaining input files to the end of the output file.</p>
OVER	<p>Writes the input files over (replaces) the existing output files on a one-for-one basis, regardless of file size. If an output file does not already exist, its corresponding input file is written to a new file with the corresponding output file name. If you specify multiple input files and a single output file, COPY appends the remaining input files to the end of the output file.</p>
AFTER	<p>Appends the input file or files to the current data in the existing output file or files. If the output file does not already exist, all listed input files will be concatenated into a new file with the listed output file name.</p>
outpath-list	<p>One or more pathnames for the output files. Multiple pathnames must be separated by commas. Separating blanks are optional. If you omit the preposition and outpath-list parameters, COPY displays the output at your console screen (TO :CO:).</p>

DESCRIPTION

The COPY command can be used to perform several different operations. Some of these include:

- Creating new files (TO preposition).

- Copying over existing files or creating new files (OVER preposition).
- Adding data to the end of existing files (AFTER preposition).
- Copying a list of files to another list of files on a one-for-one basis.
- Concatenating two or more files into a single output file.

As each file is copied, the COPY command displays one of the following messages:

<pathname>, copied TO <out-pathname>

<pathname>, copied OVER <out-pathname>

<pathname>, copied AFTER <out-pathname>

When you copy files, the number of input pathnames you specify must equal the number of output pathnames, unless you specify only one output pathname. In the latter case, COPY appends the remainder of the input files to the end of the output file. As each file is appended, the following message is displayed on the console screen:

<pathname>, copied AFTER <output-file>

If you specify multiple output files, and there are more input files than output files, or if you specify fewer input files than output files, COPY returns an error message.

Also, if you specify a wild-card character in an output pathname, you must specify the same wild-card character in the corresponding input pathname. Other combinations result in error conditions.

You cannot successfully use COPY to copy a directory to a data file or to another directory. Although a directory can be copied, the attributes of the directory are lost. That is, the directory can no longer be used as a directory. However, a file listed under one directory can be copied to another directory. For example:

```
COPY SAMP/TEST/A TO :F1:/ALPHA/BETA
```

This would copy the A data file to a different volume, directory, and filename, where the new file's pathname would be :F1:/ALPHA/BETA.

The user ID of the user who invokes the COPY command is considered the owner of new files created by COPY. Only the owner can change the access rights associated with the file (refer to the PERMIT command later in this chapter).

When COPY creates new files, it sets the access rights and list of accessors as follows:

- It sets the file for ALL access (delete, read, append, and change).
- It sets the owner as the only accessor to the file.

Refer to the PERMIT command for more information about access rights and the list of accessors.

ERROR MESSAGES

- <pathname>, output file same as input file

You attempted to copy a file to itself.

- <pathname>, UPDATE or ADD access required

Either you cannot overwrite the information in a file because you do not have update access to it, or you cannot copy information to a new file because you do not have add entry access to the file's parent directory.

CREATEDIR

This command creates one or more iRMX 86 user directories. The format is as follows:



INPUT PARAMETER

<code>inpath-list</code>	One or more pathnames of the iRMX 86 directories to be created. Multiple pathnames must be separated by commas. Embedded blanks between commas and pathnames are optional.
--------------------------	--

DESCRIPTION

CREATEDIR creates a directory with all access rights available to you, the owner. That is, you can delete, list, add, and change the contents of the directory you created with CREATEDIR. Other users (except the system manager) have no access to the directory unless you use the PERMIT command (described later in this chapter) to change the access rights and list of accessors.

The following message is displayed if a directory is successfully created:

<directory-name>, directory created

You can create new directories that are subordinate to other directories. For example:

```
CREATEDIR AB/DC/EF/GH
```

causes the newly-created directory GH to be nested within existing directory EF, which in turn, is nested within directory DC, and so on. The directories AB, DC, and EF must already exist before entering this command.

You can check the contents of the directory at any time by using the DIR command to list the directory (see the DIR command in this chapter).

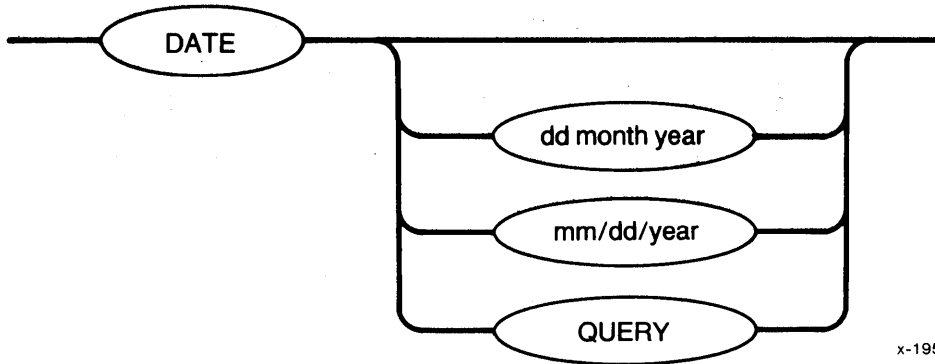
ERROR MESSAGE

- <directory-name>, file already exists

The pathname of the directory to be created already exists.

DATE

This command sets a new system date or displays the current date. The format is as follows:



INPUT PARAMETERS

- dd** Two-digit number that specifies the day of the month.
- month** Designation for the month. You can enter the whole name (such as AUGUST) or enough characters to distinguish one month from another (for example, AU, to distinguish AUGUST from APRIL). You can use this form for specifying the month only when using the "dd month year" format.
- mm** Numerical designation for the month (for example: 1 represents January, 2 represents February, etc.). You can use this form for specifying the month only when using the "mm/dd/year" format.
- year** Designation for the year. You can enter this as a two- or four-digit number, as follows:

<u>entered year</u>	<u>actual year</u>
0 through 77	2000 through 2077
78 through 99	1978 through 1999
100 through 1977	error
1978 through 2099	1978 through 2099
2100 and up	error

- QUERY** Causes DATE to prompt for the date by issuing the following message:

DATE:

DATE continues to issue this prompt until you enter a valid date.

DESCRIPTION

If you set one date parameter, you must set all three; there are no default settings for individual date parameters. You must separate the dd, month, and year entries with single blanks.

If you omit the date parameters, DATE displays the current date and time in the following form:

```
dd mmm yy, hh:mm:ss
```

When the Operating System displays the date, it displays only the first three characters of the month and the last two digits of the year. It separates the hours, minutes, and seconds of the time with colons.

If you request the date on a non-timing system, DATE displays the following message:

```
00:00:00
```

Refer to the TIME command in this chapter if you wish to set the system clock while setting the date.

ERROR MESSAGES

- `<date>`, invalid date

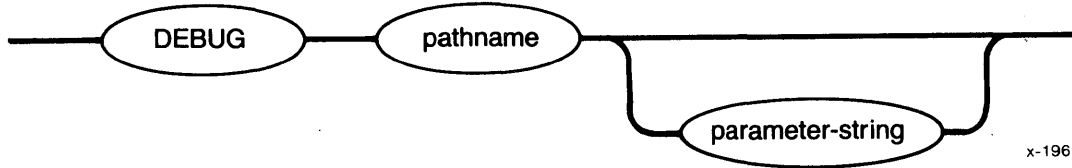
You entered an invalid date. This could result from specifying a day that is invalid for the month you specified (such as 31 FEB 82), entering characters for the year parameter that do not fall into the legitimate ranges listed under the year parameter, entering a month parameter that does not uniquely identify the month, or entering invalid characters.

- `<parameter>`, invalid syntax

You specified both a date and the QUERY parameter in the DATE command.

DEBUG

This command allows you to debug your iRMX 86 application jobs if your system is configured with the iSBC 957B package.



INPUT PARAMETERS

pathname	Pathname of the file containing the application program to be debugged.
parameter-string	String of required, optional, and default parameters that can be used in the command line to load and execute the application program.

DESCRIPTION

DEBUG loads your specified application program into main memory and transfers control to the iSBC 957B monitor. You can then use the iSBC 957B monitor to single-step, display registers, and set breakpoints within the program. Refer to the USER'S GUIDE FOR THE iSBC 957B iAPX 86, 88 INTERFACE AND EXECUTION PACKAGE for a complete description of the iSBC 957B functions.

When you invoke the DEBUG command, it displays the following message:

```
DEBUG file, <pathname>
```

where <pathname> is the pathname of the file containing the application job to debug. Then DEBUG loads the application job and displays information about the location of the job's segments and groups. Figure 3-1 shows an example of this output.

As Figure 3-1 shows, the first line of the display lists the token for the job that was created. The remaining lines list the base portions of all segments and groups assigned by LINK86 when the code was linked. The S(n) and G(n) values are the same as those that appear on the link map. Therefore, you can match the base values shown in this display with the offset values shown in the link map to determine the exact location of a symbol listed in the link map. Refer to the iAPX 86, 88 FAMILY UTILITIES USER'S GUIDE for information about LINK86 and the link map.

SEGMENT AND GROUP MAP FOR JOB: A88F

NAME	BASE	NAME	BASE	NAME	BASE	NAME	BASE	NAME	BASE
S(1)	9E4E	S(2)	9E32	S(3)	9CFF	S(5)	9CEC	S(6)	A863
S(7)	A229	S(8)	A84D	S(9)	A152	S(13)	9C91	S(15)	9C85
S(17)	9C67	S(18)	9C5C						
G(1)	A229	G(2)	A152						

Figure 3-1. Sample DEBUG Display

When DEBUG executes, the iSBC 957B package disables interrupts. This causes the time-keeping function to stop when code is not executing. This slowing of the timing function:

- Affects the ability of the Nucleus to execute time-out tasks that have provided time limits to system calls, such as RECEIVE\$UNITS and RECEIVE\$MESSAGE.
- Affects the ability of the Basic I/O System to keep track of the time-of-day and write its data structures to secondary storage.

Unless you use the monitor's NQ command to single-step through code, the iSBC 957B package cannot tolerate interrupts while single-stepping. The NQ command disables interrupts while single-stepping, allowing you to single-step through code without being interrupted by the system clock.

When DEBUG is invoked to debug an application program, it loads the application program into its own dynamic memory. This means that the application program obtains dynamic memory from the memory pool of DEBUG, not from the memory pool of the user session. Therefore, programs that experience problems with insufficient memory when run independently might not experience those problems when run under the control of DEBUG.

ERROR MESSAGE

- <exception value> : <exception mnemonic> command aborted by EH

While processing, the DEBUG command encountered an exceptional condition. Therefore, the Human Interface's exception handler aborted the command. The message lists the exception code that occurred.

DELETE

This command removes data files and empty directories from secondary storage. The format is as follows:



INPUT PARAMETERS

- inpath-list** One or more pathnames for the named data files or empty directories to be deleted. Multiple pathname entries must be separated by commas. Separating blanks are optional.
- QUERY** Causes the DELETE command to ask for your permission to delete each file in the list. Prior to deleting a file, the DELETE command displays the following query:

<pathname>, DELETE?

Enter one of the following (followed by a carriage return) in response to the query:

<u>Entry</u>	<u>Action</u>
Y or y	Delete the file.
E or e	Exit from DELETE command.
R or r	Continue deleting without further query.
Any other character	Do not delete file; query for next file in sequence.

DESCRIPTION

The DELETE command allows you to release unused secondary storage space for new uses by removing empty directories and unneeded data files. To delete a file, you need not be the owner of the file; however you must have delete access to the file. If a user or program is accessing the file (has a connection to the file) when you enter the DELETE command, DELETE marks the file for deletion and deletes it when all connections to the file are gone.

Non-empty directories cannot be deleted. If you wish to delete a directory that contains files, you must first delete all its contents. For example, if you wish to delete a directory named ALPHA whose entire contents consist of a directory BETA containing a data file SAMP, you would enter the following command:

```
DELETE ALPHA/BETA/SAMP, ALPHA/BETA, ALPHA
```

This would delete all the files contained under ALPHA before deleting the directory itself.

DELETE displays the following message as it deletes each file or marks the file for deletion:

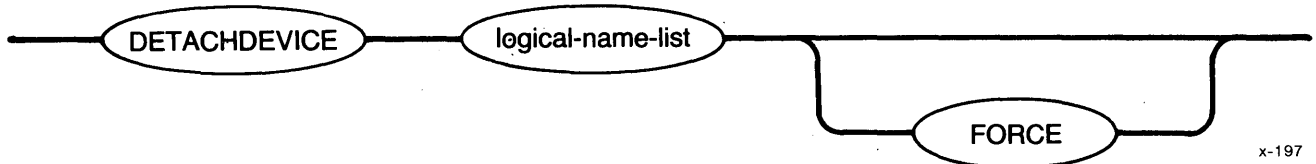
```
<pathname>, DELETED
```

ERROR MESSAGES

- <pathname>, DELETE access required
You do not have permission to delete the specified file.

DETACHDEVICE

This command detaches the specified devices and deletes their logical names from the root job's object directory. The format of this command is as follows:



INPUT PARAMETER

logical-name-list	One or more logical names of the physical devices that are to be detached. Colons surrounding each logical name are optional; however, if you use colons, you must use matching colons. Multiple logical names must be separated by commas.
FORCE	Causes DETACHDEVICE to detach the device even if connections to files on the device currently exist.

DESCRIPTION

The DETACHDEVICE command allows you to detach a device without having to reconfigure the system. After a device is detached, no volume mounted on that device is accessible for system use.

Unless you are the system manager (user ID 0), you can detach only the following devices:

- Devices that are configured with your user ID as the owner ID
- Devices you originally attached using the ATTACHDEVICE command
- Devices originally attached using the WORLD parameter of ATTACHDEVICE
- Devices originally attached by user WORLD (user ID 65535)

DETACHDEVICE returns an error message if you attempt to detach devices originally attached by other users. This prevents users from detaching devices belonging to other users and from accidentally detaching system volumes. However, the system manager can detach all devices.

Unless you specify the FORCE parameter, you cannot detach a device if any connections exist to files on the device (that is, if other users are currently accessing the device). However, the FORCE parameter causes DETACHDEVICE to delete all connections to files on the device before detaching the device.

After detaching the device and deleting its logical name from the root job's object directory, the DETACHDEVICE command displays the following message:

<logical-name>, detached

NOTE

Using the DETACHDEVICE command to detach the device containing your Human Interface commands causes loss of access to Human Interface functions until the system is restarted.

ERROR MESSAGES

- <logical name>, can't detach device
<logical name>, <exception value> : <exception mnemonic>

An exceptional condition occurred which prevented DETACHDEVICE from detaching the device. This message lists the resulting exception code.

- <logical name>, device does not belong to you

The device was originally attached by a user other than WORLD or you. Thus you cannot detach the device.

- <logical name>, device has outstanding file connections

There are existing connections to files on the device. Because you did not specify the FORCE parameter, DETACHDEVICE does not detach the device.

- <logical name>, device is in use

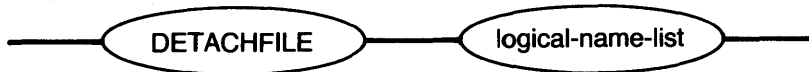
Another user or program is accessing the device (has a connection to a file). Therefore, you must specify the FORCE parameter in order to detach the device.

- <logical name>, outstanding connections to device have been deleted

There were outstanding connections to files on the volume. However, because you specified the FORCE parameter, DETACHDEVICE deleted those connections. This is a warning message that does not prevent DETACHDEVICE from detaching the device.

DETACHFILE

This command allows you to terminate the association of a logical name with a file. The format of this command is as follows:



x-198

PARAMETERS

logical-name-list List of logical names, separated by commas, that represent the files to be detached. Each logical name must be contain 1 to 12 characters. Colons surrounding each logical name are optional; however, if you use colons, you must use matching colons.

DESCRIPTION

You establish an association between a file and a logical name by entering the ATTACHFILE command. DETACHFILE breaks this association. It does this by uncataloging the logical name from your interactive job's global object directory. When DETACHFILE detaches a file in this manner, it displays the following message:

<logical name>, detached

where <logical name> is the name you specified.

You cannot use DETACHFILE to detach logical names that do not represent files. DETACHFILE returns an error message if you make such an attempt. In particular, you cannot use DETACHFILE to detach devices.

You cannot use DETACHFILE to detach logical names originally created by other users. DETACHFILE searches for logical names in the global object directory of your interactive job only. It does not search the root job's object directory nor the object directories of any other interactive jobs.

ERROR MESSAGES

- <exception value> : <exception mnemonic> invalid global job

The Human Interface encountered an internal system problem when it attempted to remove the logical name from the global job's object directory. The message lists the resulting exception code.

- <logical name>, logical name does not exist

The logical name is not cataloged in the global object directory of your interactive job.

- <logical name>, logical name not allowed

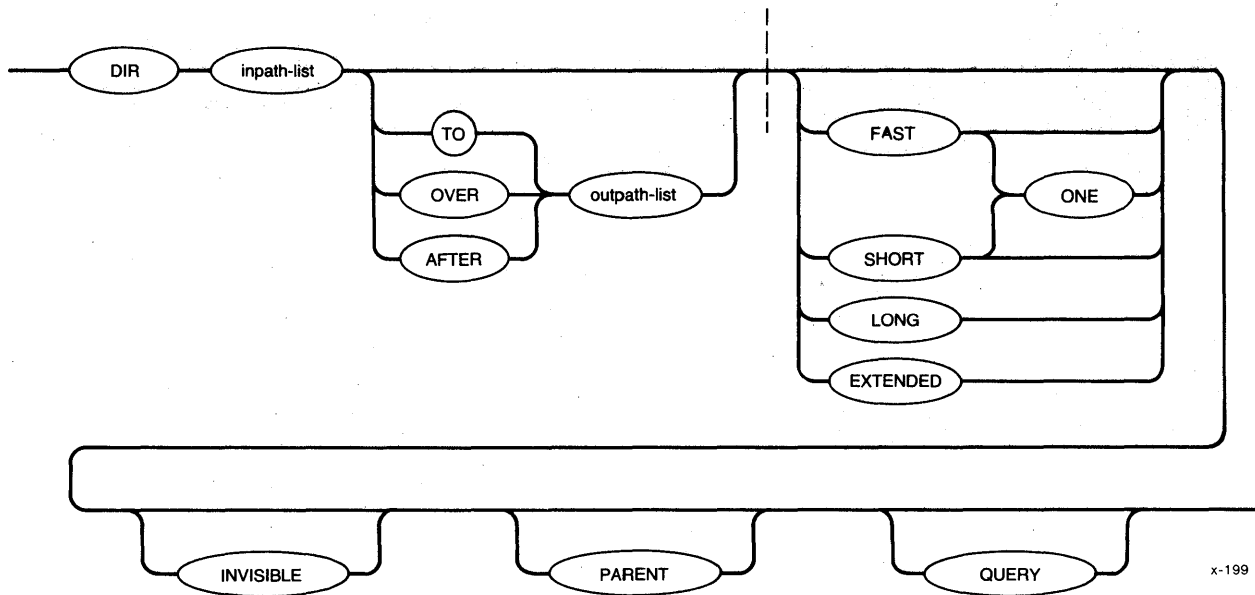
The logical name you specified was either :\$, :HOME:, :CI:, or :CO:. You cannot detach the files associated with these logical names.

- <logical name>, not a file connection

The logical name you specified is cataloged in the global object directory of your interactive job, but it is not the logical name of a file.

DIR

This command lists the names and attributes of the data and directory files contained in a given directory. The format of the command is as follows:



INPUT PARAMETERS

- inpath-list** One or more pathnames of the directories to be listed (the pathnames can represent data files if the PARENT parameter is also specified). Multiple directory pathname entries must be separated by commas. Separating blanks are optional. If no pathname is specified, the user's default directory is listed.
- FAST** Lists only the filenames and directory names in the directory. The output format contains five columns of filenames unless you also specify the ONE parameter (see Figure 3-2 at the end of this command description). FAST is the default if you omit the listing format.
- SHORT** Lists the file information in a two-column format (see Figure 3-3 at the end of this command description).
- ONE** Lists the output of a FAST or SHORT listing in single-column format. ONE is the default number of columns for EXTENDED or LONG listings.

LONG Lists file information in a one-line format (see Figure 3-4 at the end of this command description).

EXTENDED Lists all available information for each data file or directory file in the directory. The first line for each file is the same as for the LONG form. The second line contains the last access date, creation date, and the accessor list. The listing is in a double-column format (see Figure 3-5 at the end of this command description).

INVISIBLE Lists the invisible files (those beginning with the characters "R?" or "r?") in addition to the rest of the files in the directory. If you omit this parameter, DIR does not display invisible files.

PARENT Causes DIR to display an entry for the directory specified in the inpath-list in addition to the files contained in the directory. This parameter is useful for obtaining information about the root directory of a volume when using the LONG or EXTENDED parameters.

QUERY Causes the DIR command to prompt you for permission to list a directory by issuing the following message:

<pathname>, DIR?

Enter one of the following (followed by a carriage return) in response to the query:

<u>Entry</u>	<u>Action</u>
Y or y	List the directory.
E or e	Exit from DIR command.
R or r	Continue listing directories without further query.
Any other character	Do not list directory; query for the next directory, if any.

OUTPUT PARAMETERS

TO Copies the directory listing to the specified destination data file. If the destination file already exists, DIR displays the following information:

<pathname>, already exists, OVERWRITE?

Enter Y, y, R, or r if you wish to delete the existing file. Enter any other character if you do not wish to delete the file.

If you omit the TO/OVER/AFTER preposition and the output pathname, TO :CO: is the default.

OVER	Copies the directory listing to the specified output file and writes over (replaces) the previous contents.
AFTER	Appends the directory listing to the current contents of the specified output file.
outpath-list	One or more pathnames of the files to receive the directory listing. Multiple pathname entries must be separated by commas. Separating blanks are optional. If you omit the preposition and the outpath-list, the default destination is the user's console screen (TO :CO:).

DESCRIPTION

You do not need to be the owner of a directory to list its contents with DIR; however, you must have list access to the directory.

The amount of information listed for each file depends upon what listing format you specify (FAST, SHORT, LONG, or EXTENDED) in the DIR command. An example of each type of listing format is provided at the end of the DIR command description in Figures 3-2 through 3-5 respectively. Table 3-3, which follows the figures, provides an explanation of the illustrated headings.

If you want to list the default user directory but also wish to specify a listing format other than FAST, use the default directory name explicitly. For example:

```
DIR :$: EXTENDED
```

displays a listing of the default directory in the EXTENDED format. Note that the identity of your default directory is a configuration option.

Figures 3-2, 3-3, 3-4, and 3-5 show output examples for FAST, SHORT, LONG, and EXTENDED listing formats respectively. Table 3-3 defines the displayed column headings.

If a file name begins with the characters "R?" or "r?", it is an invisible file. Normally DIR does not display invisible files. However, you can specify the INVISIBLE parameter to display these files.

-DIR alpha

```

03 MAR 82  04:25:40
  DIRECTORY OF alpha  ON mvol
fname1 fname2 fname3 fname4 fname5
fname6 fname7 fname8 fname9 fname10
fnamell . . .
.
.
.

```

Figure 3-2. FAST Directory Listing Example (Default Listing Format)

-DIR mydirectory2 S

```

03 MAR 82  21:55:24
  DIRECTORY OF mydirectory2 ON mvol
  NAME          AT  ACC  BLKS  LENGTH  NAME          AT  ACC  BLKS  LENGTH
append          -R--   02  1425   alpha.obj      DRAU   3   2871
REFERENCE       DR  -L--   1   10     DATA         DR  DLAC   1    4
LEMONADEIT      DRAU 123456789 123456789
time           DRAU   6  5374   detachdevice   DRAU   4   3414
test           -R--   5  4415   schedule       ---U   7   6976
testprog.a86   -RA-   2  2040   DATABASE.LST   -RAU  11  10336
EXPERIMENTAL   DR  -LAC   1   20     BACKUP        DR  DLAC   1    10

      13 FILES      44 BLOCKS      36895 BYTES

```

Figure 3-3. SHORT Directory Listing Example

-DIR mydirectory1 L

03 MAR 82 21:55:24

DIRECTORY OF mydirectory1 ON myvol

NAME	AT	ACC	BLKS	LENGTH	GRAN		OWNER	LAST MOD
					VOL	FIL		
ed		-R--	11	1057	1024	1	# 47	02 MAR 82
programs	DR	DL--	30	30185	1024	1	# 47	03 MAR 82
fmat		DRAU	1	39	1024	1	# 655535	08 NOV 81
OBJFILE		---U	3	2895	1024	1	# 47	18 DEC 81
ALPHA1.P86		DLAC	2	1304	1024	1	# 50	22 OCT 81
ALPHA1.MPI		DLAC	6	5397	1024	1	# 50	22 OCT 81
manuals	DR	-L--	1	304	1024	1	# 47	02 JUL 80
7 FILES			54 BLOCKS	41181 BYTES				

Figure 3-4. LONG Directory Listing Example

-DIR mydir E

03 MAR 82 21:55:24

DIRECTORY OF mydir ON myvol

NAME	AT	ACC	BLKS	LENGTH	GRAN		OWNER	LAST MOD		
					VOL	FIL				
programs	DR	DL--	30	30185	1024	1	# 47	03 MAR 82		
								CREATION: 01 JAN 81 04:05:44	ACCESSORS	ACC
								LAST ACC: 03 MAR 82 05:52:33	# 47	DL--
								LAST MOD: 03 MAR 82 05:52:33	# 50	-LA-
ed		-R--	11	1057	1024	1	# 47	02 MAR 82		
								CREATION: 11 NOV 81 12:24:05	ACCESSORS	ACC
								LAST ACC: 02 MAR 82 14:22:16	# 47	-R--
								LAST MOD: 02 MAR 82 14:22:16	# 82	-L--
fmat		DRAU	1	39	1024	1	# 65535	08 NOV 81		
								CREATION: 01 NOV 81 08:54:39	ACCESSORS	ACC
								LAST ACC: 03 MAR 82 14:56:59	# 65535	DRAU
								LAST MOD: 08 NOV 81 20:44:01		
testdir	DR	DLAC	1	32	1024	1	# 47	01 MAR 82		
								CREATION: 02 FEB 82 15:02:42	ACCESSORS	ACC
								LAST ACC: 03 MAR 82 09:32:53	# 47	DLAC
								LAST MOD: 01 MAR 82 13:13:07	# 50	-LA-
							# 65535	-L--		
4 FILES			43 BLOCKS	32213 BYTES						

Figure 3-5. EXTENDED Directory Listing Example

Table 3-3. Directory Listing Headings

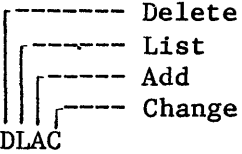
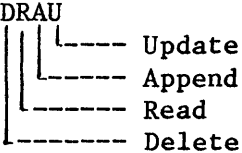
Heading	Meaning
NAME	14-character file name.
AT	File attribute, where: DR = Directory MP = Bit map file blank = Data file
ACC	File access rights of the user who entered the DIR command, where: <div style="display: flex; align-items: center; margin-left: 40px;"> Directories:  </div> <div style="display: flex; align-items: center; margin-left: 40px;"> Data Files:  </div>
BLKS	Nine-digit number (five digits on SHORT listing) giving the volume-granularity units allocated to the file. On the SHORT display, if the number of digits exceeds five, DIR displays the file in the nine-digit form (see the LEMONADEIT file in Figure 3-5).
LENGTH	10-digit number (7 digits on SHORT listing) giving the length of the file in bytes. On the SHORT form, if the number of digits exceeds 7, the file is displayed in the 10-digit form (see the LEMONADIT file in Figure 3-5).
VOL	Five-digit number giving the volume granularity in bytes.
FIL	Three-digit number giving the granularity of the file in multiples of volume granularity.
OWNER	14-character, alphanumeric owner name.
LAST MOD	Date of last file modification.
LAST ACC	Date of last file access.
CREATION	Date of file creation.

Table 3-3. Directory Listing Headings (continued)

Heading	Meaning
ACCESSORS	User IDs of users who have access to the file.
ACC	Access rights of the corresponding user. The format of this field is identical to ACC as described previously.

ERROR MESSAGES

- no directory files found

None of the files you specified were directories.

- <pathname>, READ access required

You do not have read (list) access to the directory.

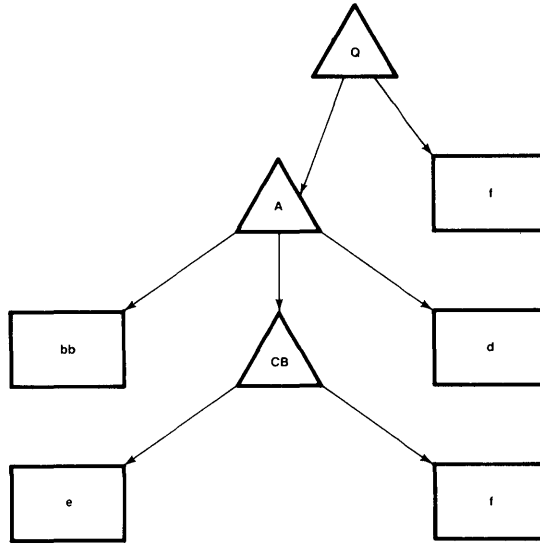
- <pathname>, UPDATE or ADD access required

Either you cannot overwrite the information in a file because you do not have update access to it, or you cannot copy information to a new file because you do not have add entry access to the file's parent directory.

EXAMPLES

The examples that follow show how a directory's files are listed when you use your default prefix in a directory's pathname. In the examples, directory names are enclosed in triangles; data file names are enclosed in rectangles.

Assume you have the following directory structure for your files:



Example 1:

Suppose your default prefix is :F0:Q. This example shows the files that would be listed in response to various DIR commands. It shows the pathnames that you could enter and the resulting files that DIR would list.

<u>Pathname</u>	<u>Files Listed</u>
omitted	A, f
f	not allowed because f is a data file
A	bb, CB, d
A/d	not allowed because d is a data file
A/CB	e, f
A/CB/e	not allowed because e is a data file

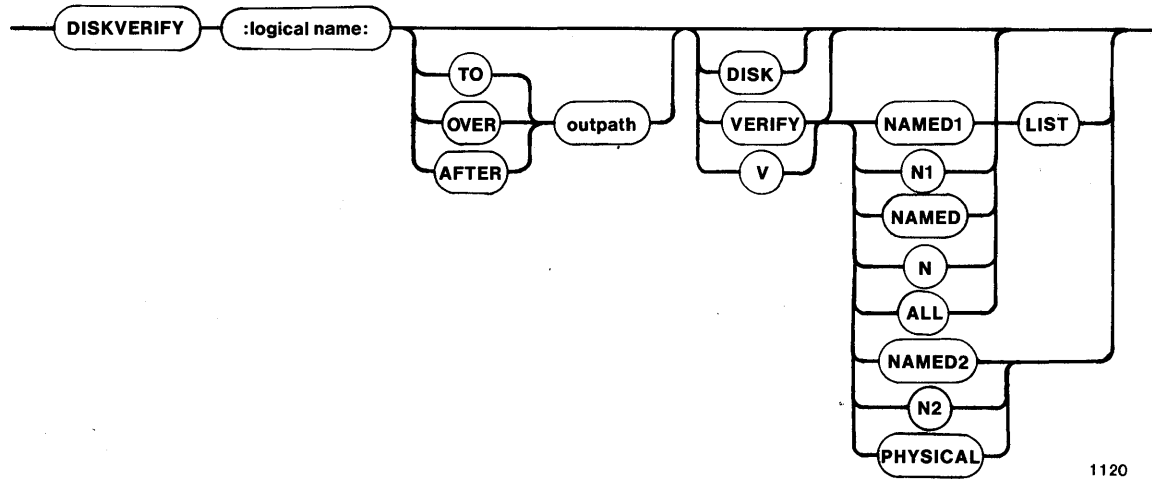
Example 2:

Suppose your default prefix is :F0:Q/A. This example also shows the files that would be listed in response to various DIR commands.

<u>Pathname</u>	<u>Files Listed</u>
omitted	bb, CB, d
A	not allowed because directory A does not contain an entry A
CB	e, f

DISKVERIFY

This command invokes the disk verification utility which verifies the data structures of iRMX 86 physical and named volumes. This utility can also be used to reconstruct portions of the volume and perform absolute editing on the volume. The format of the DISKVERIFY command is as follows:



1120

INPUT PARAMETERS

:logical-name: Logical name of the secondary storage device containing the volume.

DISK Displays the attributes of the volume (such as type of volume, device granularity, block size, number of blocks, interleave factor, extension size, volume size, and number of fnodes) and returns control to you at the Human Interface level. You can then enter any Human Interface command.

If you omit this parameter (and the VERIFY parameter), the utility displays a sign-on message and the utility prompt (*). You can then enter individual disk verification commands. These commands are described in the iRMX 86 DISK VERIFICATION UTILITY REFERENCE MANUAL.

- VERIFY or V Performs a verification of the volume. If you specify this parameter and omit the options, the utility performs the NAMED verification.
- If you specify this parameter, the utility performs the verification function and returns control to you at the Human Interface level. You can then enter any Human Interface command.
- If you omit this parameter (and the DISK parameter), the utility displays a sign-on message and the utility prompt (*). You can then enter individual disk verification commands. These commands are described in the IRMX 86 DISK VERIFICATION UTILITY REFERENCE MANUAL.
- NAMED1 or N1 VERIFY option that applies to named volumes only. This option checks the fnodes of the volume to ensure that they match the directories in terms of file type and file hierarchy. (Refer to the description of the FORMAT command for more information about fnodes.) This option also checks the information in each fnode to ensure that it is consistent. As a result of this option, DISKVERIFY displays a list of all files on the volume that are in error, with information about each file. Refer to the IRMX 86 DISK VERIFICATION UTILITY REFERENCE MANUAL for more information.
- NAMED or N VERIFY option that performs both the NAMED1 and NAMED2 verification functions on a named volume. If you omit the VERIFY option, NAMED is the default option.
- ALL VERIFY option that applies to both named and physical volumes. For named volumes, this option performs both the NAMED and PHYSICAL verification functions. For physical volumes, this option performs only the PHYSICAL verification function.
- NAMED2 or N2 VERIFY option that applies to named volumes only. This option checks the allocation of fnodes on the volume, checks the allocation of space on the volume, and verifies that the fnodes point to the correct locations on the volume. Refer to the IRMX 86 DISK VERIFICATION UTILITY REFERENCE MANUAL for more information.
- PHYSICAL VERIFY option that applies to both named and physical volumes. This option reads all blocks on the volume and checks for I/O errors.

LIST VERIFY option that you can use with other VERIFY options that, either explicitly or implicitly, specify the NAMED1 option. When you use this option, the file information generated by VERIFY is displayed for every file on the volume, even if the file contains no errors. Refer to the iRMX 86 DISK VERIFICATION UTILITY REFERENCE MANUAL for more information.

OUTPUT PARAMETERS

TO Copies the output from the disk verification utility to the specified file. If the file already exists, DISKVERIFY displays the following information:

<pathname>, already exists, OVERWRITE?

Enter Y, y, R, or r to write over the existing file. Enter any other character if you do not wish to overwrite the file.

If no preposition is specified, TO :CO: is the default.

OVER Copies the output from the disk verification utility over the specified file.

AFTER Appends the output from the disk verification utility to the end of the specified file.

outpath Pathname of the file to receive the output from the disk verification utility. If you omit this parameter and the TO/OVER/AFTER preposition, the utility copies the output to the console screen (TO :CO:). You cannot direct the output to a file on the volume being verified. If you attempt this, the utility returns an E\$NOT_CONNECTED error message.

DESCRIPTION

When you enter the DISKVERIFY command, the utility responds by displaying the following line:

iRMX 86 DISK VERIFY UTILITY, Vx.y

where Vx.y is the version number of the utility. If you specify the VERIFY or DISK parameter in the DISKVERIFY command, the utility performs the operation specified in the parameter and copies the output to the console (or to the file specified by the outpath parameter).

Refer to the iRMX 86 DISK VERIFICATION UTILITY REFERENCE MANUAL for a description of the output. After generating the output, the utility returns control to the Human Interface, which prompts you for more Human Interface commands. The following is an example of a DISKVERIFY command that uses the VERIFY option:

```
-DISKVERIFY :F1: VERIFY NAMED2
iRMX 86 DISK VERIFY UTILITY , Vx.y
DEVICE NAME = F1          : DEVICE SIZE = 0003E900 : BLOCK SIZE = 0080

'NAMED2' VERIFICATION
  BIT MAPS O.K.
```

The following is an example of a DISKVERIFY command that uses the DISK option:

```
-DISKVERIFY :F2: DISK
iRMX 86 DISK VERIFY UTILITY, Vx.y
Device name = WFO
  Named disk, Volume name = UTILS
    Device gran = 0080
    Block size = 0080
    No of blocks = 0000072D : No of Free blocks = 00000408
    Volume size = 0003E900
    Interleave = 0005
  Extension size = 03
    No of fnodes = 0038      : No of Free fnodes = 0022
```

However, if you omit the VERIFY and DISK parameters from the DISKVERIFY command, the utility does not return control to the Human Interface. Instead, it issues an asterisk (*) as a prompt and waits for you to enter individual DISKVERIFY commands. The following is an example of such a DISKVERIFY command:

```
-DISKVERIFY :F1:
*
```

After you receive the asterisk prompt, you can enter any of the DISKVERIFY commands listed in the iRMX 86 DISK VERIFICATION UTILITY REFERENCE MANUAL.

ERROR MESSAGES

- argument error

The VERIFY option you specified is not valid.

- command syntax error

You made a syntax error when entering the command.

- device size inconsistent
size in volume label = <value1> : computed size = <value2>

When the disk verification utility computed the size of the volume, the size it computed did not match the information recorded in the iRMX 86 volume label. It is likely that the volume label contains invalid or corrupted information. This error is not a fatal error, but it is an indication that further error conditions may result during the verification session. You may have to reformat the volume or use the disk verification utility to modify the volume label. Refer to the iRMX 86 DISK VERIFICATION UTILITY REFERENCE MANUAL for more information about the disk verification utility commands.

- not a named disk

You tried to perform a NAMED, NAMED1, or NAMED2 verification on a physical volume.

The NAMED1, NAMED2, and PHYSICAL verification options can also produce error messages. Refer to the iRMX 86 DISK VERIFICATION UTILITY REFERENCE MANUAL for more information about these messages.

EXAMPLE

The following command performs both named and physical verification of a named volume.

-DISKVERIFY :F1: VERIFY ALL

iRMX 86 DISK VERIFY UTILITY, Vx.y

DEVICE NAME = F1 : DEVICE SIZE = 0003E900 : BLK SIZE = 0080

'NAMED1' VERIFICATION

'NAMED2' VERIFICATION

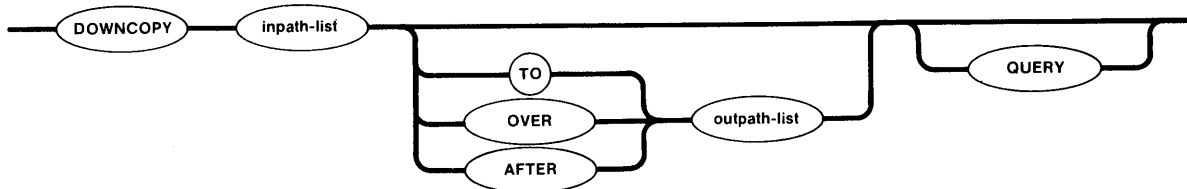
BIT MAPS O.K.

'PHYSICAL' VERIFICATION

NO ERRORS

DOWNCOPY

This command copies files from a volume on an iRMX 86 secondary storage device to a volume on an ISIS-II secondary storage device via the iSBC 957B Interface and Execution package. The format is as follows:



INPUT PARAMETERS

inpath-list

One or more iRMX 86 pathnames for files, separated by commas, that are to be copied to ISIS-II secondary storage. Separating blanks between pathnames are optional. The files may be copied in the listed sequence either on a one-for-one basis or concatenated into one or more files.

QUERY

Causes the Human Interface to prompt for permission to copy each iRMX 86 file to the listed ISIS-II destination file. Depending on which preposition you specify (TO, OVER, or AFTER), the Human Interface prompts with one of the following queries:

<pathname>, copy down TO <outfile>?

<pathname>, copy down OVER <outfile>?

<pathname>, copy down AFTER <outfile>?

Enter one of the following in response to the query:

<u>Entry</u>	<u>Action</u>
Y or y	Copy the file.
E or e	Exit from the DOWNCOPY command.
R or r	Continue copying files without further query.
Any other character	Do not copy this file; query for the next file in sequence.

OUTPUT PARAMETERS

- TO** Reads iRMX 86 files and copies them TO new ISIS-II files in the listed sequence. If the specified output files already exist in the ISIS-II directory when the TO parameter is used, DOWNCOPY displays the following message:
- <filename>, already exists, OVERWRITE?
- Enter Y, y, R, or r if you wish to delete the existing file. Enter any other character if you do not wish the existing file to be deleted.
- If no preposition is specified, TO :CO: (ISIS-II console screen) is the default. If more input files than output files are specified, the remaining input files are appended to the end of the last-specified ISIS-II file.
- OVER** Copies the iRMX 86 input files OVER the existing ISIS-II destination files in the specified sequence. If you specify multiple input files and one output file, DOWNCOPY appends the remaining input files to the end of the output file.
- AFTER** Copies the iRMX 86 input files, in sequence, AFTER the end of data on the existing ISIS-II destination files.
- outfile-list** One or more ISIS-II filenames for the output files. Multiple filenames must be separated by commas. Separating blanks are optional. If the preposition and output file defaults are used in the command line, the output goes to the ISIS-II console screen.

DESCRIPTION

The DOWNCOPY command cannot be used to copy directories from an iRMX 86 system to a Series III microcomputer development system; only files can be copied.

Before you enter a DOWNCOPY command on the iRMX 86 console keyboard, your target system must be connected to a Series III system via the iSBC 957B package, and the package must be running. To do this, you must start your iRMX 86 system from the Series III terminal (either by loading the software into the target system and using the monitor G command to start execution, or by using the monitor B command to bootstrap load the software). DOWNCOPY does not function if you start up your system from the iRMX 86 terminal or if you establish the link between the Series III system and target system after starting up your iRMX 86 system.

When DOWNCOPY copies files to the development system, it turns off all ISIS-II file attributes.

As each file in the input list is copied, one of the following messages will be displayed on the Human Interface console output device (:CO:):

<pathname>, copied down TO <out-filename>

<pathname>, copied down OVER <out-filename>

<pathname>, copied down AFTER <out-filename>

When the DOWNCOPY command is executing, the iSBC 957B package disables interrupts. This affects services such as the time-of-day clock. Also, the Operating System is unable to receive any characters that you type-ahead while the DOWNCOPY command is executing.

ERROR MESSAGES

- <pathname>, DELETE access required

DOWNCOPY could not replace an existing ISIS-II file because the file is write-protected.

- <pathname>, ISIS ERROR: <nnn>

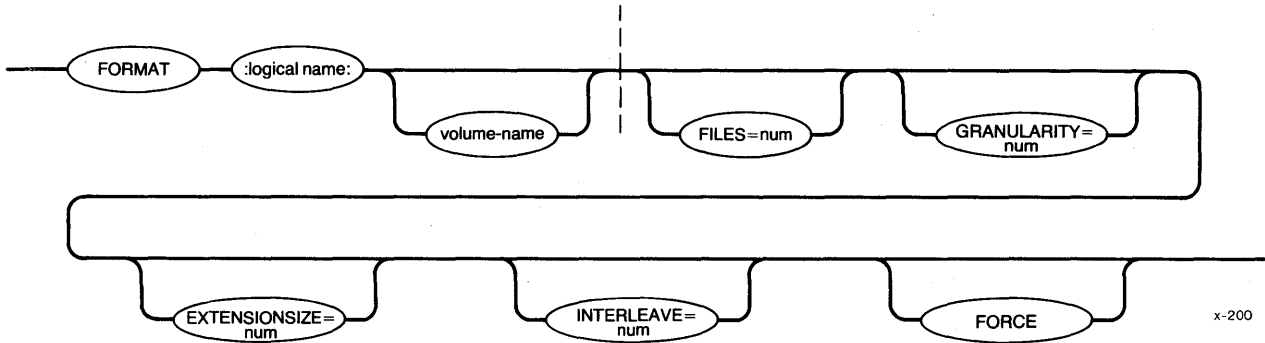
An ISIS-II Operating System error occurred when DOWNCOPY tried to transfer the file to the Microcomputer Development System. Refer to the INTELLEC SERIES III MICROCOMPUTER DEVELOPMENT SYSTEM CONSOLE OPERATING INSTRUCTIONS for a description of the resulting error code.

- ISIS link not present

The the iRMX 86 system is not connected to the development system via the iSBC 957B package.

FORMAT

This command formats or reformats a volume on an iRMX 86 secondary storage device, such as a diskette, hard disk, or bubble memory. The format is as follows:



x-200

INPUT PARAMETERS

- :logical-name:** Logical name of the physical device-unit to be formatted. You must surround the logical name with colons. Also, you must not leave space between the logical name and the succeeding volume name parameter.
- volume-name** Six-character, alphanumeric ASCII name, without embedded blanks, to be assigned to the volume. If you include this parameter, you must not leave spaces between the logical name and the volume name.
- FILES=num** Defines the maximum decimal number of user files that can be created on a NAMED volume. (This parameter is not meaningful when formatting a PHYSICAL volume and is ignored if specified for such volumes.) FORMAT uses the information specified in this parameter to determine how many structures (called *fnodes*) to create on the NAMED volume. The range for the FILES parameter is 1 through 32,761, although the maximum number of user files you can define depends on the settings of the GRANULARITY and EXTENSIONSIZE parameters (as explained in the "Description" portion of this command write-up). When you use this parameter, FORMAT creates six additional *fnodes* for internal system files. If not specified, the default is 50 user files.

FORCE Forcibly deletes any existing connections to files on the volume before formatting the volume. If you do not specify **FORCE**, you cannot format the volume if any connections to files on the volume still exist.

GRANULARITY=num Volume granularity; the minimum number of bytes to be allocated for each increment of file size on a **NAMED** volume. (This parameter is not meaningful for **PHYSICAL** volumes, and is ignored if specified for such volumes.) **FORMAT** rounds the value you specify up to the next multiple of the device granularity. Then it places the decimal number in the header of the volume, where it becomes the default file granularity when a file is created on the volume. The range is 1 through 65,535 (decimal) bytes, although the maximum allowable volume granularity depends on the settings of the **FILES** and **EXTENSIONSIZE** parameters (as explained in the "Description" portion of this write-up). If not specified, the default granularity is the device granularity. Once the volume granularity is defined, it applies to every file created on that volume.

NOTE

Using a large volume granularity (in excess of 1024), might cause users to exceed their memory limits when executing programs that reside on the volume. This can occur because the Operating System uses the volume granularity as a minimum buffer size when reading and writing files.

EXTENSIONSIZE=num Size, in bytes, of the extension data portion of each file. (This parameter is not meaningful for **PHYSICAL** volumes, and is ignored if specified for such volumes.) The range is 0 through 255 (decimal), although the maximum allowable extension size depends on the settings of the **FILES** and **GRANULARITY** parameters (as explained in the "Description" portion of this write-up). If not specified, the default extension size is 3 bytes.

INTERLEAVE=num Interleave factor for a **NAMED** or **PHYSICAL** volume. Acceptable values are 1 through 255 decimal. If not specified, the default value is 5. See the interleave discussion under "Description" in this command write-up.

NAMED

The volume can store only named files; that is, the volume can hold many files (up to the number of fnodes allocated), each of which can be accessed by its pathname. A diskette or hard disk surface are examples of devices that would be formatted for named files. If neither NAMED nor PHYSICAL is specified, the volume is formatted for the file specified when you attached the device (with the ATTACHDEVICE command).

PHYSICAL

The volume can be used only as a single, physical file. The GRANULARITY and FILES parameters are not meaningful when PHYSICAL is specified for the volume. If neither NAMED nor PHYSICAL is specified, the volume is formatted for the file type specified when you attached the device (with the ATTACHDEVICE command).

DESCRIPTION

Every physical device-unit used for secondary storage must be formatted before it can be used for storing and then accessing its files. For example, every time you mount a previously unused diskette into a drive, you must enter a FORMAT command to format that diskette as a new volume before you can create, store and access files on it.

Once a volume is formatted, its name becomes a volume identifier when you display the root directory of the volume, and the name appears in the directory's heading. Although the Human Interface uses the volume name in its own internal processing when you access the volume, you need not specify the volume name in any subsequent command after the volume is formatted. You must specify only the logical name of the secondary storage device that contains the volume.

Volume Name

The Human Interface requires a volume name for its own internal processing of your read/write accesses to the volume. Once the volume is formatted, you need never specify the volume name in a command; you only specify the logical name for the device on which you later mount the volume.

For diskettes, a volume name gives you a method for identifying a volume in case the stick-on label on the diskette gets lost or destroyed. You need only mount the disk on a drive and enter a DIR command for that drive to get a directory listing that specifies the volume name.

Fnodes

The number of fnodes on a volume defines the number of files that can exist on the volume. You can specify the number of fnodes reserved for user files with the FILES parameter. Each fnode is a data structure that contains information about a file. Each time you create a file on the volume, the Operating System records information about the file in an unused fnode. Later, it uses the fnode to determine the location of the file on the volume.

Internal Files

When you format a named volume, FORMAT creates six internal system files. It names three of these files and lists their names in the root directory of the volume. The files are:

<u>file</u>	<u>description</u>
R?SPACEMAP	Volume free space map
R?FNODEMAP	Free fnodes map
R?BADBLOCKMAP	Bad blocks map

It grants the user WORLD read access to these files. Refer to the iRMX 86 DISK VERIFICATION UTILITY REFERENCE MANUAL for more information about these files.

Root Directory

FORMAT also uses one of the fnodes for the root directory. It lists the user who formats the volume as the owner, giving that user all access rights. No other user has access to the root directory until the owner explicitly grants access. The owner can grant other users access to the volume via the PERMIT command described later in this chapter. However, because the owner has all access rights to the root directory, the owner can obtain exclusive access to the volume, and can obtain delete access to any file created on the volume, even files created by other users.

Extension Data

Each fnode contains a field that stores extension data for its associated file. An operating system extension can access and modify this extension data by invoking the A\$GET\$EXTENSION\$DATA and A\$SET\$EXTENSION\$DATA system calls (refer to the iRMX 86 BASIC I/O SYSTEM REFERENCE MANUAL for more information). When you format a volume, you can use the EXTENSIONSIZE parameter to set the size of the extension data field in each fnode. Although you can specify any size from 0 to 255 bytes, the Human Interface requires all fnodes to have at least 2 bytes of extension data.

Volume Granularity

The default volume granularity is always the granularity of the physical device for the volume. For example, if the default granularity for a device is 128 bytes of secondary storage, the I/O System will automatically allocate permanent storage to each new file you create on that volume in multiples of 128 bytes, regardless of whether the file requires the full amount.

Relationship between FILES, GRANULARITY, and EXTENSIONSIZE

Although the FILES, GRANULARITY, and EXTENSIONSIZE parameters have maximum values which are listed in the parameter descriptions, the combination of these parameters must also satisfy the following formula:

$$(87 + \text{EXTENSIONSIZE}) \times (\text{FILES} + 6) / \text{GRANULARITY} \leq 65535$$

where all numbers are decimal. FORMAT displays an error message if the combination of parameter values exceeds the limit.

Interleave Factor

The interleave factor applies to volumes formatted either for NAMED or PHYSICAL files. The interleave factor specifies the logical sector sequence. The interleave specification maximizes access speed for the files on a given volume, depending on the intent of volume and the device configuration. For example, an interleave factor of 5 for a flexible disk drive means that, for each file, the I/O System reads and writes every fifth sector on the diskette, starting with an index of 1. (Other, hard disk systems may be different, depending on your configuration.) With the appropriate interleave factor, the I/O System does not need to wait for the disk to make a complete revolution before it accesses the next sector; the next sector by an increment of 5 is ready to be accessed for read/write by the time the previously accessed sector has been processed.

Output Display

The FORMAT command displays one of the following message when volume formatting is completed. For physical volumes:

```

volume (<volume name>) will be formatted as a PHYSICAL volume
  device gran.           = <number>
  interleave            = <number>
  volume size           = <k-number> K (or M)
volume formatted

```

For named volumes:

```

volume (<volume name>) will be formatted as a NAMED volume
  granularity           = <number>           sides       = <sides>
  interleave           = <number>           density      = <density>
  files                 = <number>           disk size    = <d-size>
  extensionsize        = <number>
  volume size          = <k-number> K (or M)
volume formatted

```

where:

<volume name>	Volume name specified in the FORMAT command.
<number>	Decimal number as specified in the command (or the default)
<k-number>	Volume size in K (1024-byte units) or M (1048576-byte units). FORMAT displays the volume size in Kbyte units unless the size is greater than 25 Mbytes.
<sides>	Number of sides of the volume that will be formatted (1 or 2). This field is displayed only for flexible diskettes in which FORMAT can recognize this characteristic.
<density>	Density at which the volume will be formatted (single or double). This field is displayed only for flexible diskettes in which FORMAT can recognize this characteristic.
<d-size>	Size of the volume (8 or 5.25). This field is displayed only for flexible diskettes in which FORMAT can recognize this characteristic.

ERROR MESSAGES

- <logical name>, can't attach device
<logical name>, <exception value> : <exception mnemonic>

FORMAT cannot attach the device for formatting, or it cannot re-attach the device (that is, restore it to its original condition) after formatting takes place.

- <logical name>, can't detach device
<logical name>, <exception value> : <exception mnemonic>

FORMAT cannot detach the device for formatting, which means that the volume does not exist, the volume is busy, or the device on which the volume is mounted is not currently attached to the system.

FORMAT

- <logical name>, device is in use

You cannot format the volume because there are outstanding connections to files on the volume and you did not specify the FORCE parameter.

- <vol-name>, fnode file size exceeds 65535 volume blocks

The values you specified for fnode size, granularity, and extension data size cause the formula listed in the "Description" section to exceed its limit.

- <number>, invalid number

You specified an out-of-range number for any of the FILES, GRANULARITY, EXTENSIONSIZ, or INTERLEAVE parameters.

- <logical name>, outstanding connections to device have been deleted

There were outstanding connections to files on the volume. However, because you specified the FORCE parameter, FORMAT deleted those connections. This is a warning message that does not prevent FORMAT from formatting the volume.

- 0085 : E\$LIST, too many values

You entered multiple logical-name/volume-name combinations separated by commas. FORMAT can format only one volume per invocation.

- <volume name>, volume name is too long

FORMAT requires the volume name you specify to be 6 characters or less.

INITSTATUS

This command displays the initialization status of Human Interface terminals. The format of this command is as follows:



DESCRIPTION

INITSTATUS displays at the user terminal the initialization status of all Human Interface terminals. Figure 3-6 illustrates the format of the INITSTATUS display.

TERMINAL DEVICE NAME	CONFIG EXCEP	DEVICE EXCEP	INIT EXCEP	USER STATE	JOB ID	USER ID
.T0.	0000	0000	0000	LE	1	65535
.T1.	0000	0000	0000	-E	2	1
.T3.	0000	0002		--		
.T4.	0021			--		

Figure 3-6. INITSTATUS Display

The columns listed in Figure 3-6 contain the following information.

TERMINAL DEVICE NAME	The physical name of the terminal, as defined during the configuration of the Basic I/O System and as attached by the Human Interface. Periods surround each name.
CONFIG EXCEP	Hexadecimal condition code that the Human Interface received when it attempted to interpret the terminal definition and user definition files (refer to the iRMX 86 CONFIGURATION GUIDE for more information). A zero value indicates a normal condition. Nonzero values indicate exceptional conditions. Refer to Appendix B for a list of exception codes.
DEVICE EXCEP	Hexadecimal condition code that the Human Interface received when it originally attached the terminal as a physical device.

INITSTATUS

INIT EXCEP	Condition code that the Human Interface received when it created a job for the interactive session.
USER STATE	Two characters that indicate the current state of the terminal. The first character can be either: <ul style="list-style-type: none">L The terminal is locked and cannot be reinitialized (refer to the LOCK command later in this chapter).- The terminal is unlocked. The second character can be either: <ul style="list-style-type: none">E The Human Interface created the interactive job associated with this terminal and the job exists.- The interactive job does not exist.
JOB ID	A sequential number that the Human Interface assigns to the interactive job during initialization. You must specify this number as a parameter in the JOBDELETE command in order to delete the corresponding interactive job.
USER ID	User ID associated with the interactive job. This is the identification of the user that the Human Interface associates with the job when the user begins a Human Interface session.

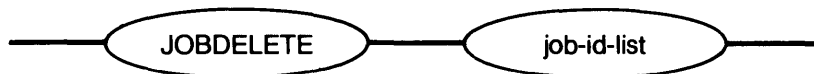
ERROR MESSAGE

- not a multi-access system

The Human Interface cannot return information about terminals because it is not configured for multi-access.

JOBDELETE

This command deletes a running interactive job. The system manager can use this command to delete any interactive job. Other users can delete only those interactive jobs that have the same user ID that they have. The format of this command is as follows:



x-202

where:

job-id-list	One or more job IDs, separated by commas, of the interactive jobs to be deleted. You can obtain the IDs of jobs by invoking the INITSTATUS command (described earlier in this chapter).
-------------	---

DESCRIPTION

The JOBDELETE command allows users to delete interactive jobs. Deleting an interactive job causes the Human Interface to terminate the corresponding user session.

When JOBDELETE attempts to delete a job, it first attempts to delete the job's offspring jobs (for example, a SUBMIT file or a program invoked as a result of an RQ\$CREATE\$IO\$JOB system call). It deletes multiple levels of offspring jobs. However, JOBDELETE cannot delete any interactive job (or offspring) that contains extension objects. Refer to the IRMX 86 NUCLEUS REFERENCE MANUAL for more information about deleting jobs containing extension objects.

Normally, when a user's interactive job is deleted, the Human Interface recreates the interactive job, thus restarting the user session. However, if the LOCK command (described later in this chapter) has been specified for the user's terminal, the Human Interface does not automatically recreate the user's interactive job after a JOBDELETE command. Therefore, the system manager can use the combination of LOCK and JOBDELETE to remove users from the system prior to a system shutdown.

As JOBDELETE deletes each job, it displays the following message at the user terminal (:CO:):

<job-ID>, deleted

where <job-ID> is the identifier of the deleted job.

ERROR MESSAGES

- <job-ID>, does not exist

The interactive job associated with the identifier <job-ID> does not exist. It has already been deleted.

- <job-ID>, invalid job id

The number <job-ID> is not a job ID that is associated with any terminal managed by the Human Interface.

- <job-ID>, job does not belong to you

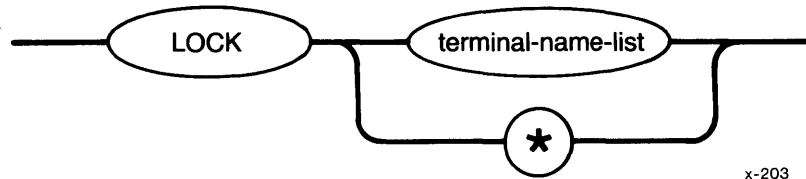
The user who attempted to delete the interactive job does not have the same user ID as the interactive job or is not the system manager.

- <job-ID>, not deleted
<job-ID>, <exception value> : <exception mnemonic>

An exceptional condition occurred, preventing JOBDELETE from deleting the job <job-ID>. JOBDELETE displays the exception code that resulted.

LOCK

This command prevents the Human Interface from automatically recreating the interactive job for a terminal once that interactive job has been deleted. This process is called locking the terminal. The system manager can use this command to lock any terminal. Other users can lock only those terminals whose interactive jobs have the same user ID that they have. The format of this command is as follows:



x-203

where:

- terminal-name- One or more terminal device names, separated by commas, of the terminals to be locked. You can obtain the terminal device names by invoking the INITSTATUS command (described earlier in this chapter).
- * A special character indicating that all configured terminals should be locked.

DESCRIPTION

The system manager can use the LOCK command in conjunction with the JOBDELETE command either to selectively delete users from the system or to shut down the entire system. LOCK prevents the Human Interface from recreating a user's interactive job once that job has been deleted. Interactive jobs can be deleted in any of the following ways:

- As a result of the JOBDELETE command (described earlier in this chapter)
- By shutting off the terminal
- By entering an end-of-file character (CTRL/z) at the terminal

As LOCK locks each terminal, it displays the following message to the user terminal (:CO:):

<terminal-name>, locked

where <terminal-name> is the terminal device name of the locked terminal.

LOCK

ERROR MESSAGES

- lock not allowed

You attempted to lock your own terminal. Only system managers can lock their own terminals.

- <terminal-name>, not found

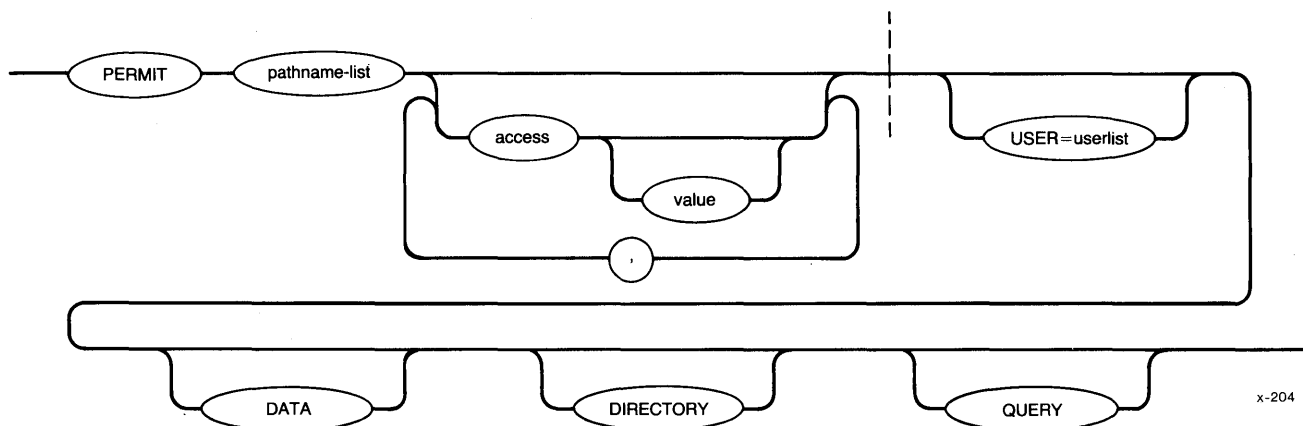
A terminal with device name <terminal-name> is not configured into your application system.

- not a multi-access system

The LOCK command does not function if the Human Interface is configured for single-access only.

PERMIT

This command allows you to grant or revoke user access to files that you own. The format of this command is as follows:



x-204

INPUT PARAMETERS

pathname-list One or more pathnames, separated by commas, of the files that are to have their access rights or list of accessors changed.

access Access characters that grant or rescind the corresponding access to the file, depending on the value parameter that follows. The possible values include:

<u>value</u>	<u>access</u>
D	Delete
L or R	List (for directories) and read (for data files)
A	Add entry (for directories) and append (for data files)
C or U	Change (for directories) and Update (for data files)
N	Rescinds all access not explicitly granted (used without an accompanying value)

PERMIT

If specified without an accompanying value, each access character grants the specified access. Specifying N alone rescinds all access and removes the users specified with the USER parameter from the file's access list. Specifying N with other characters grants the access specified by those characters and rescinds all other access. You can use L and R interchangeably for both data files and directories; likewise C and U.

value

Value which specifies whether to grant or rescind the associated access right. Possible values include:

<u>value</u>	<u>meaning</u>
0	Rescind the access right
1	Grant the access right

The default value is 1. That is, specifying an access character without a value grants the corresponding access.

user-list

User IDs for whom the previously-specified access rights apply. Two special values are also acceptable for this parameter. They are:

WORLD	Special user ID (OFFFh) giving all users access to the file.
*	Designator indicating that the access rights apply to all users currently in the file's access list.

The Operating System limits each file to three user IDs in the access list. If you omit this parameter, PERMIT assumes the user ID associated with your interactive job.

DATA

Specifies that the access information applies to the data files in the pathname list. If you omit both the DATA and DIRECTORY parameters, PERMIT assumes both.

DIRECTORY

Specifies that the access information applies to the directories in the pathname list. If you omit both the DATA and DIRECTORY parameters, PERMIT assumes both.

QUERY

Causes PERMIT to prompt for permission to modify the access rights associated with each file. It does this by displaying the following message:

<pathname>,
 accessor = <new id>, <new access>, PERMIT?--

Enter one of the following (followed by a carriage return) in response to the query:

<u>Entry</u>	<u>Action</u>
Y or y	Change the access.
E or e	Exit from the PERMIT command.
R or r	Change the access and continue with the command without further query.
Any other character	Do not change access; continue with PERMIT command and query for next access change, if any.

DESCRIPTION

You can use the PERMIT command to update the access information for the following files:

- Files for which you are listed as the owner.
- Files for which you have change-entry access to the file's parent directory.

You cannot change the access information for other files. PERMIT can perform the following functions:

- Adding or subtracting users from a file's list of accessors. This list determines which users have access to the file.
- Setting the type of access (access rights) granted to the users in the accessor list.

Currently the Operating System allows only three user IDs in the list of accessors, but one of these IDs can be the special ID WORLD, which grants access to all users.

You specify the type of access to be granted or rescinded by means of access characters and values. You can concatenate access characters and values together or you can separate the individual access specifications with commas. For example, if you want to grant delete access and rescind add and update access, you could enter any of the following combinations:

```
AODUO
AO,D,UO
AOD1UO
AO,D1,UO
```

As you can see from the previous lines, D is equivalent to D1. Also, the order in which you specify access characters is not important.

If there are multiple occurrences of an access character in the PERMIT command, PERMIT uses the last such character to determine the access. For example, the combination:

```
DO,A1,R1,D1
```

is the same as the combination:

```
A1,R1,D1
```

In the first combination, the D1 overrides the DO.

You can use the N character to rescind all access to the file. If specified alone, it removes user IDs from the accessor list. However, the N character can also be useful when changing access rights, if you don't remember the specified user's current access rights. In this case you can specify the N character first, to clear all the access rights, and follow it with other characters to grant the desired access. For example, if you want to grant list access only, instead of specifying:

```
DOAOCOL
```

you could specify:

```
NL
```

After changing the access information for a file, PERMIT displays the following information:

```
<pathname>,  
  accessor = <accessor ID>, <access>  
  .  
  .  
  .
```

where <pathname> is the pathname of the specified file, <accessor ID> is the user ID of one of the files accessors, and <access> indicates the access rights that the corresponding user has. PERMIT displays the access rights as access characters: DLAC for directories and DRAU for data files. If a particular access right is not allowed, the display replaces the corresponding character with a dash (-). For example, the display:

```
-L-C
```

indicates that the corresponding user has list and change access.

ERROR MESSAGES

- `<pathname>`, accessor limit reached

The Operating System permits only three IDs in the accessor list of a file. Before you can add another accessor, you must remove one of the current accessors by setting its access rights to N.

- `<pathname>`, directory CHANGE access required

Either you are not the owner of the file specified by `<pathname>`, or you do not have change access to the file's parent directory. You must satisfy one of these two conditions in order to use the PERMIT command.

- `<user ID>`, duplicate USER control

You must specify the keyword and parameter combination USER = userlist only once during the PERMIT command. However, you can specify multiple user IDs by separating them with commas in the userlist. PERMIT exits without updating the access rights.

- `<character>`, invalid access switch

The character you entered to indicate the access rights for the file was not a valid access character. PERMIT exits without updating the access rights.

- `<invalid id>`, invalid user id

The user IDs you supply with the USER parameter must consist of decimal or hexadecimal characters, the characters WORLD, or the character *. PERMIT exits if supplied other characters.

- missing access switches

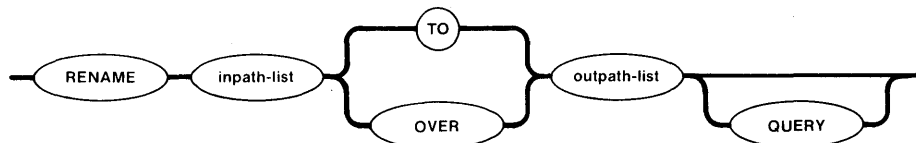
You must specify one or more access characters with the PERMIT command. PERMIT exits without updating the access rights.

- no files found

There were no files of the type you specified (data, directory, or both) in the pathname list.

RENAME

This command allows you to change the pathname of one or more data files or directories. RENAME is effective across directory boundaries on the same volume. The format is as follows:



INPUT PARAMETERS

inpath-list

One or more pathnames, separated by commas, of files or directories that are to be renamed. Blanks between pathnames are optional separators.

QUERY

Causes the Human Interface to prompt for permission to rename each pathname in the input list by issuing one of the following messages:

<oldname>, rename TO <newname>?
 <oldname>, rename OVER <newname>?

Enter one of the following (followed by a carriage return) in response to the query:

<u>Entry</u>	<u>Action</u>
Y or y	Rename the file.
E or e	Exit from RENAME command.
R or r	Continue renaming without further query.
Any other character	Do not rename file; query for the next file in sequence.

OUTPUT PARAMETERS

TO

Moves the data to the new pathnames in the output list. A new pathname in the output list should not already exist. If the output pathname already exists, RENAME displays the following message:

<pathname>, already exists, DELETE?

Enter Y, y, R, or r to delete the existing file. Enter any other character if you do not wish to delete the file. In the latter case, RENAME skips over the specified file without changing it and attempts to rename the next pathname in the list.

OVER

Changes each old pathname in a list to the corresponding new pathname, even if the new pathname already exists. OVER cannot be used to rename a non-empty directory over another non-empty directory.

outpath-list

List of new pathnames. Multiple pathnames must be separated by commas. Separating blanks are optional.

DESCRIPTION

The primary distinction between the RENAME command and the COPY command is that, as the RENAME command runs, it releases the pathnames of the input files for new uses without performing any further operation on the files.

Another distinction between RENAME and COPY is that RENAME cannot be used across volume boundaries; that is, you cannot use the RENAME command to rename a file or move data from a volume located on one secondary storage device to a volume located on another secondary storage device (for example, from one diskette to another). An attempt to do so causes an error message. Use the COPY command or a combination of COPY and DELETE commands if you wish to rename files or move data across volume boundaries.

To use RENAME, you must have delete access to the current file and add-entry access to the destination directory. If you rename a file OVER an existing file, you must also have delete access to the second file.

Although RENAME can be used to rename an existing directory pathname TO a new pathname, it cannot be used to rename an existing directory OVER another existing directory. For example:

```
-RENAME ALPHA TO DELTA          ;allowed
-RENAME ALPHA OVER BETA         ;not allowed (unless BETA is empty)
-RENAME ALPHA/SAMP1 OVER BETA/TEST1 ;allowed
```

RENAME

NOTE

Changing the name of a directory also changes the path of all files listed in that directory. All subsequent accesses to those files must specify the new pathnames for the files.

As each file in a pathname list is renamed, the RENAME command displays one of the following messages, as appropriate:

<old pathname>, renamed TO <new pathname>
or
<old pathname>, renamed OVER <new pathname>

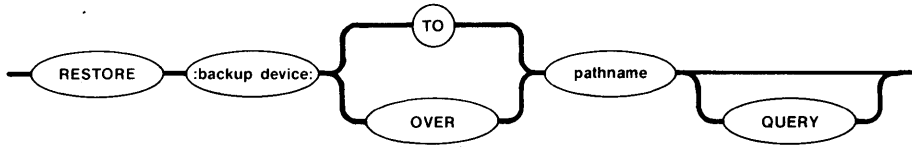
ERROR MESSAGES

- <old pathname>, DELETE access required
You cannot rename a file unless you have delete access to that file.
- <new pathname>, directory ADD ENTRY access required
You cannot rename a file unless you have add-entry access to the destination directory.
- <new pathname>, new pathname same as old pathname
You specified the same name for the input pathname as you did for the output pathname.
- TO or OVER preposition expected
Either you used the AFTER preposition with the RENAME command or the number of files in your inpath-list did not match the number in your outpath-list.

RESTORE

This command restores files to a named volume by copying them from a backup volume.

The format of this command is as follows:



INPUT PARAMETERS

:backup device: Logical name of the backup device from which RESTORE restores files.

QUERY Causes the Human Interface to prompt for permission to restore each file. The Human Interface prompts with one of the following queries:

<pathname>, RESTORE data file?

or

<pathname>, RESTORE directory?

Enter one of the following responses to the query:

<u>Entry</u>	<u>Action</u>
Y or y	Restore the file.
E or e	Exit from the RESTORE command.
R or r	Continue restoring files without further query.
Any other character	If data file, do not restore the file; if directory file, do not restore the directory or any file in that portion of the directory tree. Query for the next file, if any.

RESTORE

OUTPUT PARAMETERS

TO Restores the files from the backup volume to new files on the named volume, if the files do not already exist on the named volume. If a file being restored already exists on the named volume, RESTORE displays the following message:

<pathname>, already exists, OVERWRITE?

Enter one of the following in response to the query:

<u>Entry</u>	<u>Action</u>
Y, y, R, or r	Delete the file and replace it with the one from the backup volume.
E or e	Exit from the RESTORE command.
Any other character	Do not restore the file; go on to the next file.

OVER Restores the files from the backup volume over (replaces) the files on the named volume. If a file does not exist on the named volume, RESTORE creates a new file on the named volume.

pathname Pathname of a file which receives the restored files (you must specify a directory pathname when restoring more than one file). If you specify a logical name for a device, RESTORE places the files under the root directory for that device. However, the device must contain a volume formatted as a named volume. If you wish to restore files to the directory in which they originated, you should specify the same pathname parameter as you used with the BACKUP command.

DESCRIPTION

RESTORE is a utility which copies files from backup volumes (where the BACKUP command originally saved them) to named volumes. RESTORE copies the files to any directory you specify, maintaining the hierarchical relationships between the backed-up files.

Normally, when RESTORE copies files, it copies only those files to which you have access. When it copies these files to the named volume, it establishes your user ID as the owner ID (regardless of what the previous owner ID was). However, if you are the system manager (user ID 0), RESTORE restores all files from the backup volume and leaves the owner ID the same as it was.

When copying files, RESTORE restores the following information:

- File name
- Access list
- Extension data
- File granularity
- Contents of the file

RESTORE changes the creation, last modification, and last access dates of the file to the current date.

Each backup volume which is used as input to the RESTORE command must contain files placed there by the BACKUP command. In addition, if the backup operation required multiple backup volumes, you must restore these volumes in the same order as they were backed up.

The output volume which receives the restored files must be a named volume. You must have sufficient access rights to the files in that volume to allow RESTORE to perform all necessary operations. For RESTORE to create new files on a named volume, you must have add entry access to directories on that volume. For RESTORE to restore files over existing files, you must have add entry and change entry access to directories in that volume and delete, append, and update access to data files.

When you enter the RESTORE command, RESTORE displays the following sign-on message:

```
IRMX 86 DISK RESTORE UTILITY Vx.y
```

where Vx.y is the version number of the utility. Then it prompts you for a backup volume.

Whenever RESTORE requires a new backup volume, it issues the following message:

```
<backup device>, mount backup volume #<nn>, enter Y to continue:
```

where <backup device> indicates the logical name of the backup device and <nn> the number of the requested volume. (RESTORE in some cases displays additional information to indicate problems with the current volume.) In response to this message, place the backup volume in the backup device (make sure that the volume number is correct if the backup operation involved multiple volumes). Enter one of the following:

<u>Entry</u>	<u>Action</u>
Y, y, R, or r	Continue the restore process.
E or e	Exit from the RESTORE command.
Any other character	Invalid entry; reprompt for entry.

RESTORE

RESTORE continues prompting you until you supply the correct backup volume.

As it restores each file, RESTORE displays one of the following messages at the Human Interface console output device (:CO:):

<pathname>, restored

or

<pathname>, directory restored

ERROR MESSAGES

- <pathname>, access to directory or file denied

RESTORE could not restore a file, either because you did not have add entry access to the file's parent directory or because you did not have update access to the file. RESTORE continues with the next file.

- <backup device>, backup volume #<nn>, <date>, mounted
<backup device>, backup volume #<nn>, <date>, required

<backup device>, mount backup volume #<nn>, enter Y to continue:

RESTORE cannot continue because the backup volume you supplied is not the one that RESTORE expected. Either you supplied a volume out of order or you supplied a volume from a different backup session. RESTORE reprompts for the correct backup volume.

- <backup device>, cannot attach volume
<backup device>, <exception value> : <exception mnemonic>

<backup device>, mount backup volume #<nn>, enter Y to continue:

RESTORE cannot access the backup volume. This could be because there is no volume in the backup device or because of a hardware problem with the device. The second line of the message indicates the iRMX 86 exception code encountered. RESTORE continues to issue this message until you supply a volume that RESTORE can access.

- <pathname>, <exception value> : <exception mnemonic>, error during BACKUP, file not restored

When the BACKUP utility saved files, it encountered an error when attempting to save the file indicated by this pathname. RESTORE is unable to restore this file. The message lists the iRMX 86 exception code encountered.

- `<pathname>, <exception value> : <exception mnemonic>, error during BACKUP, restore incomplete`

When the BACKUP utility saved the files, it encountered an error when attempting to save the file indicated by this pathname. RESTORE restores as much of the file as possible to the named volume. The message lists the iRMX 86 exception code encountered.

- `<backup device>, error reading backup volume
<backup device>, <exception value> : <exception mnemonic>`

RESTORE tried to read the backup volume but encountered an error condition, possibly because of a faulty area on the volume. The second line of the message indicates the iRMX 86 exception code encountered.

- `<pathname>, <exception value> : <exception mnemonic>, error writing output file, restore incomplete`

RESTORE encountered an error while writing a file to the named volume. This message lists the iRMX 86 exception code encountered. RESTORE writes as much of the file as possible to the named volume.

- `<pathname>, extension data not restored, <nn> bytes required`

The amount of space available on the named volume for extension data is not sufficient to contain all the extension data associated with the specified file. The value `<nn>` indicates the number of bytes required to contain all the extension data. This message indicates that the named volume on which RESTORE is restoring files is formatted differently than the named volume which originally contained the files. To ensure that you restore all the extension data from the backup volume, you should restore the files to a volume formatted with an extension size set equal to the largest value reported in any message of this kind. Refer to the description of the FORMAT command for information about setting the extension size.

- `<backup device>, invalid backup device`

The logical name you specified for the backup device was not a logical name for a device.

- `<backup device>, not a backup volume`

`<backup device>, mount backup volume #<nn>, enter Y to continue:`

The volume you supplied on the backup device was not a backup volume. RESTORE continues to issue this message until you supply a backup volume.

RESTORE

- <pathname>, not restored

For some reason, RESTORE was unable to restore a file from the backup volume. RESTORE continues with the next file. Another message usually precedes this message to indicate the reason for not restoring the file.

- output specification missing

You did not specify a pathname to indicate the destination of the restored files.

- <pathname>, READ access required

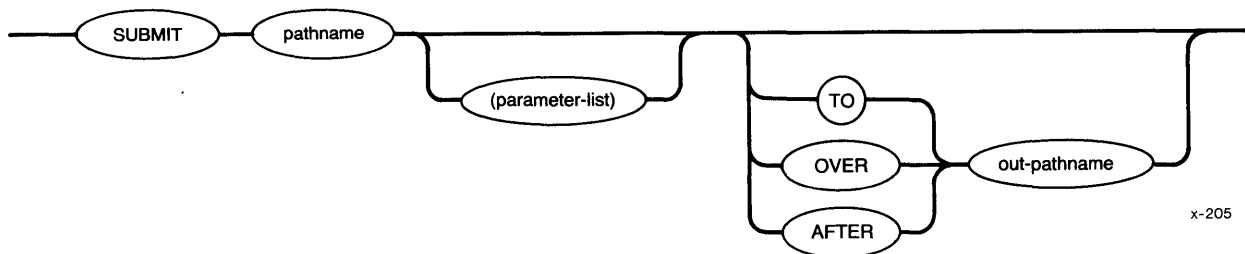
You do not have read access to a file on the backup volume; therefore RESTORE cannot restore the file.

- <pathname>, too many input pathnames

You attempted to enter a list of logical names as logical names for backup devices. You can enter only one input logical name per invocation of RESTORE.

SUBMIT

This command reads and executes a set of commands from a file in secondary storage instead of from the console keyboard. To use the SUBMIT command you must first create a data file that defines the command sequence and formal parameters (if any). The format of the command is as follows:



INPUT PARAMETERS

- pathname** Name of the file from which the commands will be read. This file may contain nested SUBMIT files.
- parameter-list** Actual parameters that are to replace the formal parameters in the SUBMIT file. You must surround this parameter list with parentheses. You can specify as many as 10 parameters, separated by commas, in the SUBMIT command. If you omit a parameter, you must reserve its position by entering a comma. If a parameter contains a comma, space, or parenthesis, you must enclose the parameter in single quotes. The sum of all characters in the parameter list must not exceed 512 characters.

OUTPUT PARAMETERS

- TO** Causes the output from each command in the SUBMIT file to be written to the specified new file instead of the console screen. If the output file already exists, the SUBMIT command displays the following message:

<pathname>, already exists OVERWRITE?

Enter Y, y, R, or r if you wish the existing output file to be deleted. Enter any other character if you do not wish the existing file to be deleted. A response other than Y or y causes the SUBMIT command to be terminated and you will be prompted for a new command entry.

OVER Causes the output for each command in the SUBMIT file to be written over the specified existing file instead of the console screen.

AFTER Causes the output from each command in the SUBMIT file to be written to the end of an existing file instead of the console screen.

out-pathname Pathname of the file to receive the processed output from each command executed from the SUBMIT file. If no preposition or output file is specified, TO :CO: is the default.

DESCRIPTION

Any program that reads its commands from the console input (:CI:) can be executed from a SUBMIT file. If another SUBMIT command is itself used in a SUBMIT file, it causes another SUBMIT file to be invoked. You can nest SUBMIT files to any level of nesting until memory is exhausted (each level of SUBMIT requires approximately 10K of dynamic memory). When one nested SUBMIT file completes execution, it returns control to the next higher level of SUBMIT file.

If, during the execution of SUBMIT (or any nested SUBMIT), you enter the CTRL/c character to abort processing, all SUBMIT processing exits and control returns to your user session.

When you create a SUBMIT file, you indicate formal parameters by specifying the characters %n, where n ranges from 0 through 9. When SUBMIT executes the file, it replaces the formal parameters with the actual parameters listed in the SUBMIT command (the first parameter replaces all instances of %0, the second parameter replaces all instances of %1, and so forth). If the actual parameter is surrounded by quotes, SUBMIT removes the quotes before performing the substitution. If there is no actual parameter that corresponds to a formal parameter, SUBMIT replaces the formal parameter with a null string.

When you specify a preposition and output file (other than :CO:) in a SUBMIT command, only your SUBMIT command entry will be echoed on the console screen; the individual command entries in the submit file are not displayed on the screen as they are loaded and executed.

The SUBMIT command will display the following message when all commands in the submit file have been executed:

END SUBMIT <pathname>

ERROR MESSAGES

- <pathname>, end of file reached before end of command
 The last command in the input file was not specified completely.
 For example, the last line might contain a continuation character.
- <parameter>, incorrectly formed parameter
 You separated the individual parameters in the parameter list
 with a separator character other than a comma.
- <pathname>, output file same as input file
 You attempted to place the output from SUBMIT into the input file.
- <pathname>, too many input files
 You specified more than one pathname as input to SUBMIT. SUBMIT
 can process only one file per invocation.
- <parameter>, too many parameters
 You specified more than 10 parameters in your parameter list.
- <pathname>, UPDATE or ADD access required
 SUBMIT cannot write its output to the output file because you do
 not have update access to the file (if it already exists) or
 because you do not have add access to the file's parent directory
 (if the file does not currently exist).

EXAMPLE

This example shows a SUBMIT file that uses formal parameters and the command that you can enter to invoke this SUBMIT file. The SUBMIT file, which resides on file :F1:MOVE\$FILE, contains the following lines:

```
ATTACHDEVICE F1 AS %0
CREATEDIR %0/%1
UPCOPY :F1:%2 TO %0%1/%2
```

The SUBMIT file contains three formal parameters, indicated by %0, %1, and %2. The %0 indicates the logical name of an iRMX 86 device; the %1 indicates the name of a directory on that device; the %2 indicates the name of a file which will be copied from an ISIS-II disk to the iRMX 86 device.

SUBMIT

The SUBMIT command used to invoke this file is as follows:

```
-SUBMIT :F0:MOVE$FILE (:F1:, PROG, FILE1)
```

The command sequence created and executed by SUBMIT is shown as it would be echoed on the console output device.

```
-ATTACHDEVICE F1 AS :F1:  
F1, attached as :F1:  
-CREATEDIR :F1:/PROG  
:F1:PROG, directory created  
-UPCOPY :F1:FILE1 TO :F1:PROG/FILE1  
:F1:FILE1 upcopied TO :F1:PROG/FILE1  
END SUBMIT :F0:MOVE$FILE  
-
```

SUPER

This command allows operators who are designated as system managers to change their user IDs to the system manager user ID (user ID 0). Having entered the SUPER command, these users can invoke a sub-command to change to any other user ID. The format of this command is as follows:



x-206

DESCRIPTION

SUPER allows you to change your user ID to that of the system manager. It has two sub-commands (CHANGEID and EXIT) that are available only after you have invoked SUPER. CHANGEID allows you to change your user ID to any possible value. EXIT exits the SUPER utility.

In order to invoke SUPER, you must know a password associated with the system manager. This password is stored in the user definition file for user ID 0 (refer to the IRMX 86 CONFIGURATION GUIDE for more information). After you enter the SUPER command, SUPER prompts for the password by displaying:

ENTER PASSWORD:

You must then enter the correct password. (SUPER does not echo your input at the terminal.) After you enter the correct password, SUPER changes your user ID to user ID 0 and issues the following prompt.

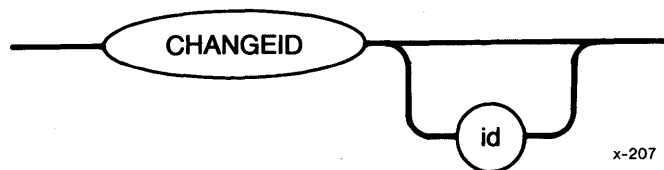
super-

This prompt is a new system prompt (replacing the "-") that appears whenever the Human Interface is ready to accept input. At this point, you can enter any Human Interface commands and access any files available to the system manager. If you create new files, they will be listed as owned by user ID 0. You can also invoke the sub-commands available with SUPER.

SUBCOMMANDS

There are two sub-commands available with SUPER: CHANGEID and EXIT. You can invoke these sub-commands only after first invoking the SUPER command.

The CHANGEID sub-command allows you to change your current user ID to any value between 0 and 65535 decimal. The format of the CHANGEID sub-command is as follows:



where:

`id` Value to which you want to change your user ID. This can be any numeric value from 0 to 65535 decimal, or the characters "WORLD" which specifies ID 65535 decimal. If you omit this value, CHANGEID sets your user ID to that of the system manager (user ID 0).

If you change your user ID to anything other than that of the system manager (user ID 0), the system prompt changes to the following:

`super(id)-`

where `id` is the decimal equivalent of your new user ID (or the characters "WORLD").

The EXIT sub-command exits from the SUPER utility. The format of this sub-command is as follows:



After you enter this sub-command, the Human Interface changes your user ID back to the ID you had before entering the SUPER command. It also changes the system prompt back to the "-" value. To change your user ID again, you must invoke the SUPER command.

ERROR MESSAGES

- `<exception value>` : `<exception mnemonic>` cannot set default user

An internal system problem prevented the Human Interface from changing your user ID.

- `<user-id>`, invalid user id

The user ID you specified contained invalid characters or was not in the range 0 to 65535 decimal.

- invalid password

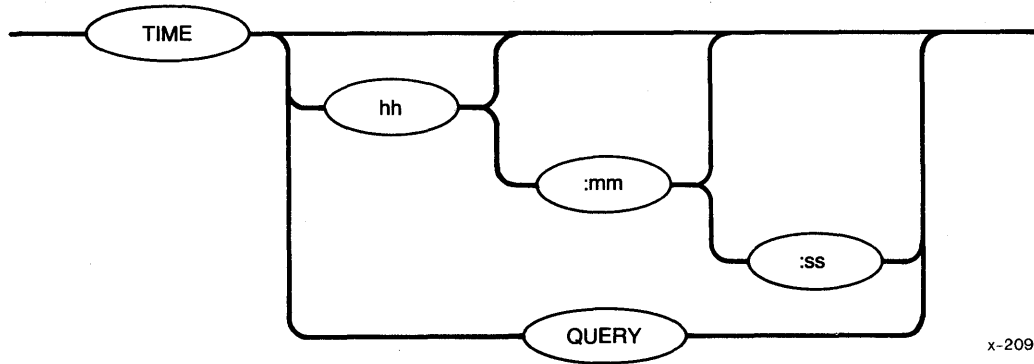
The password you entered does not match the password associated with the system manager that is listed in the user definition file.

- <exception value> : <exception mnemonic>, SUPER is un-available

The Human Interface encountered an error while reading the password you entered or while accessing the system manager's user definition file (to determine if the password is correct). This message lists the exception code that occurs.

TIME

This command sets the system clock. If no new time is entered, the TIME displays the current system time. The format is as follows:



INPUT PARAMETERS

- hh Hours specified as 0 through 24.
- mm Minutes specified as 0 through 59. If you omit this parameter, 0 is assumed.
- ss Seconds specified as 0 through 59. If you omit this parameter, 0 is assumed.
- QUERY Causes TIME to prompt you for the time by issuing the following message:

TIME:

TIME continues to issue this message until you enter a valid time.

DESCRIPTION

You must separate the individual time parameters with colons.

If you omit the time parameters, TIME displays the current date and time in the following format:

dd mmm yy, hh:mm:ss

where dd mmm yy indicates the date and hh:mm:ss indicates the time.

In order to obtain the correct time when you enter the TIME command without parameters, you must initially set the time. If you request the time on a system in which you haven't already set the time (or on a non-timing system), TIME command displays the following message:

00:00:00

See also the DATE command in this chapter if you wish to set the date in conjunction with the system clock.

ERROR MESSAGES

- <time>, invalid time

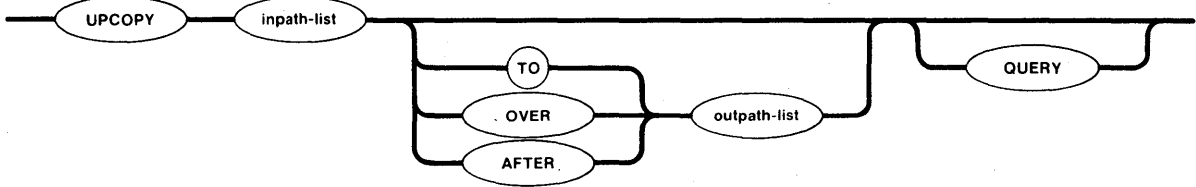
You specified an invalid or out-of-range entry for one or more of the time parameters.

- <parameter>, invalid syntax

You specified both a time and the QUERY parameter in the TIME command.

UPCOPY

This command copies files from a volume on ISIS-II secondary storage to a volume on iRMX 86 secondary storage via the iSBC 957B Interface and Execution package.



INPUT PARAMETERS

inpath-list List of one or more filenames of the ISIS-II files that are to be copied to iRMX 86 secondary storage, either on a one-for-one basis or concatenated into one or more iRMX 86 output files.

QUERY Causes the Human Interface to prompt for permission to copy each ISIS-II file to the listed iRMX 86 output file. Depending on which preposition you specify (TO, OVER, or AFTER), the Human Interface prompts with one of the following queries:

<in-pathname>, copy up TO <out-pathname>?

<in-pathname>, copy up OVER <out-pathname>?

<in-pathname>, copy up AFTER <out-pathname>?

Enter one of the following (followed by a carriage return) in response to the query:

<u>Entry</u>	<u>Action</u>
Y or y	Copy the file.
E or e	Exit from the UPCOPY command.
R or r	Continue copying files without further query.
Any other character	Do not copy this file; go to the next file in sequence.

OUTPUT PARAMETERS

TO	<p>Copies the ISIS-II file or files TO a new iRMX 86 file or files in the listed sequence. If the output file already exists, UPCOPY displays the the following message:</p> <p style="padding-left: 40px;"><pathname>, already exists, OVERWRITE?</p> <p>Enter Y, y, R, or r if you wish to write over the existing file. Enter any other character if you do not wish the file to be overwritten.</p> <p>If no preposition is specified, TO :CO: is the default. If more input files than output files are specified in the command line, the remaining input files will be appended to the end of the last listed output file.</p>
OVER	<p>Copies the listed ISIS-II input file or files OVER existing iRMX 86 destination files in the listed sequence. If more input files than output files are listed in the command line, the remaining input files will be appended to the end of the last listed output file.</p>
AFTER	<p>Appends the listed ISIS-II input file or files AFTER the end-of-data on an existing iRMX 86 output file or files in the listed sequence.</p>
outpath-list	<p>One or more pathnames of the iRMX 86 destination files. Multiple pathnames must be separated by commas. Separating blanks are optional. If the preposition and output parameter defaults are used in the command line, the output will go to the iRMX 86 console screen.</p>

DESCRIPTION

Before you enter an UPCOPY command on the iRMX 86 console keyboard, you must have your target system connected to a development system via the iSBC 957B package, and the package must be running. To do this, you must start your iRMX 86 system from the development system terminal (either by loading the software into the target system and using the monitor G command to start execution, or by using the monitor B command to bootstrap load the software). UPCOPY does not function if you start up your system from the iRMX 86 terminal or if you establish the link between development system and target system after starting up your iRMX 86 system.

The user ID of the user who invokes the UPCOPY command is considered the owner of new files created by UPCOPY. Only the owner can change the access rights associated with the file (refer to the PERMIT command).

UPCOPY

As it copies each ISIS-II file in the input list, UPCOPY displays one of the following messages at the terminal, as appropriate:

<in-pathname>, copied up TO <out-pathname>

<in-pathname>, copied up OVER <out-pathname>

<in-pathname>, copied up AFTER <out-pathname>

When the UPCOPY command is executing, the iSBC 957B package disables interrupts. This affects services such as the time-of-day clock. Also, the Operating System is unable to receive any characters that you type-ahead while the UPCOPY command is executing.

ERROR MESSAGES

- <pathname>, ISIS ERROR: <nnn>

An ISIS-II Operating System error occurred when UPCOPY tried to transfer the file to the Microcomputer Development System. Refer to the INTELLEC SERIES III MICROCOMPUTER DEVELOPMENT SYSTEM CONSOLE OPERATING INSTRUCTIONS for a description of the resulting error code.

- ISIS link not present

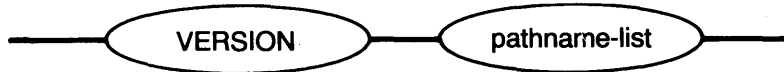
The the iRMX 86 system is not connected to the development system via the iSBC 957B package.

- <pathname>, UPDATE or ADD access required

Either you cannot overwrite the information in a file because you do not have update access to it, or you cannot copy information to a new file because you do not have add entry access to the file's parent directory.

VERSION

This command displays the version number of a command, if that command has a version number. The format of this command is as follows:



x-210

INPUT PARAMETER

`pathname-list` One or more pathnames, separated by commas, of commands for which a version number is desired.

DESCRIPTION

When you enter the VERSION command, it displays the version number of each command, if there is one, in the following format:

`<pathname>, <command-name> version is x.y`

where:

`<pathname>` Pathname of the file containing the command.

`<command-name>` Name of the specified command; Intel-supplied commands have names as listed in this manual.

`x.y` Version number of the command.

You can use VERSION to determine the version number of any Human Interface command. You can also use it to determine the version numbers of commands that you write. However, for VERSION to work on your commands, you must include a literal string in the command's source code to specify the name of the command and its version. The string must contain the following information:

```
'program_version_number=xxxx',
'program_name=yyyy...yyy',0
```

where:

`program_version_number=` You must specify this portion exactly as shown (lower case, underscore separating the words, no spaces).

VERSION

xxxx	Version number of the product. This can be any four characters, but it must be exactly four characters long.
program_name=	This portion is optional. However, if you want VERSION to recognize and display the program name, you must specify this portion exactly as shown.
yyyy...yyy	Name of the command. This can be any number of characters.
0	The literal string must be terminated with a byte of binary zero.

An example of such a literal string is:

```
DECLARE version (*) BYTE DATA('program_version_number=V8.5',  
                                'program_name=MYPROGRAM',0);
```

If your program includes this declaration, when you invoke VERSION, it will display the following information:

```
<pathname>, MYPROG version is V8.5
```

A literal string that does not include the program name is:

```
DECLARE vers2(*) BYTE DATA('program_version_number=1983',0);
```

If your program includes this declaration, when you invoke VERSION, it will display the following information:

```
<pathname>, version is 1983
```

CHAPTER 4. HUMAN INTERFACE EXAMPLES

This chapter shows you how to use some of the Human Interface commands. Its primary intent is to introduce you to basic techniques by presenting a series of examples that illustrate typical command entries.

COMMAND EXAMPLES FORMAT

To make it easier to follow the interactive dialog between the user and Human Interface in the examples, the user keyboard entries are underscored. All other items displayed in the examples are Human Interface command output. For instance, in the example:

```
-copy samp to test
samp copied TO test
-copy test
aaaaa
bbbbbb
test copied TO :CO:
-
```

the underscored items are user command entries; all other characters and lines are output by the Human Interface or the supplied commands.

Control characters, such as (CTRL/z), are enclosed in parentheses in the examples to indicate that such entries are not echoed at the console screen as they are entered. Do not actually enclose control key entries in parentheses.

HOW TO BEGIN A CONSOLE SESSION

You can begin an interactive dialog with the Human Interface after the initial program displays a sign-on message at your console screen. Although the sign-on message is a system configuration option, the message supplied with the default initial program of the Human Interface is as follows:

```
IRMX 86 HI CLI Vx.x: USER = userid
-
```

This message tells you the Human Interface is running; it also tells you your user ID. The hyphen (-) is a Human Interface prompt to indicate that it is ready to accept your first command line. Begin entering a command immediately after and on the same line as the prompt. For example:

```
-copy :ci: to test1
```

HUMAN INTERFACE EXAMPLES

HOW TO CREATE A SIMPLE DATA FILE

You can use the COPY command to create data files during a console session. Assume you wish to create a file called ALPHA and write two lines of data into the file. Also assume you wish the data file to be listed under your default directory. Enter the following command and data:

```
-copy :ci: to alpha  
aaaaa  
bbbbbb  
(CTRL/z)
```

```
:ci: copied TO alpha  
-
```

In this example, the :ci: in the COPY command line tells the command to read data from the keyboard (:ci: = console input) and write the data (aaaaa and bbbbbb) to a new file named ALPHA. Because you did not preface the file name with a directory name, COPY places the file ALPHA in your default directory.

The command does not prompt you for the data lines; you simply begin entering data after you press RETURN at the end of the command line. Your CTRL/z entry writes an end-of-file mark at the end of your data to inform the COPY command that there is no more data to be copied.

Note that after you enter the last line of data, you must press the RETURN key before you enter a CTRL/z to insert an end-of-file. Otherwise, none of the characters entered after you press the RETURN key and before you enter a CTRL/z are written to the file. For example:

```
-copy :ci: to alpha  
cccccc  
dddddd (CTRL/z) (then press RETURN)
```

would only write the data ccccc to the new file named ALPHA.

Since control characters are not echoed on the screen as you enter them, (such as a RETURN or CTRL function), the above file creation sequence would be displayed on the screen as follows:

```
-copy :ci: to alpha  
cccccc  
dddddd
```

```
:ci: copied TO alpha  
-
```

Now, assume that when you entered the COPY command line, the Human Interface sent you the following message and query:

```
-copy :ci: to alpha  
alpha, already exists, OVERWRITE?
```

HUMAN INTERFACE EXAMPLES

Whenever you create a new data file, the COPY command expects a new pathname rather than one already listed in the directory file. If your entry to the query is:

```
alpha, already exists, OVERWRITE? y
```

the COPY command deletes the data in the existing file and waits for you to enter new data under that pathname.

If your response to the query is:

```
alpha, already exists, OVERWRITE? n (or any other character except  
y)
```

-

your COPY command is ignored and the Human Interface prompts for a new command entry by issuing a hyphen (-).

HOW TO COPY FILES

COPY command options provide a number of different ways for you to copy existing files. You exercise these options either by specifying one of the TO/OVER/AFTER prepositions, by the way in which you specify your input file and output file pathname lists, or by a combination of both techniques. The services of the COPY command include:

- Copying files on a one-for-one basis.
- Displaying the contents of files at the console screen.
- Creating multiple copies of the same file.
- Copying data from multiple files to a new or existing file.
- Replacing data in one file with data from another file.
- Adding data from one or more files to the end of the data in another file.
- Combining one-for-one file copying with file concatenation in a single COPY command.

The examples that follow show you how to use these services. They also call your attention to certain file handling considerations when using the COPY command.

HUMAN INTERFACE EXAMPLES

HOW TO COPY TO NEW FILES

Copying existing files to new files is most frequently done on a one-for-one basis; that is, you list a number of existing files to be copied and a matching list of files to receive the copies. The files are copied in the same sequence you specify in the input list and output list on the command line. For example, assume you wished to copy files A1, A2, and A3 to files B1, B2, and B3 respectively. Enter the following command:

```
-copy a1, a2, a3 to b1, b2, b3  
a1 copied TO b1  
a2 copied TO b2  
a3 copied TO b3  
-
```

You could also make use of the wild card feature when copying files. If the files A1, A2, and A3 are the only files in the directory that begin with the character "A", you can use the following command to perform the same operation:

```
-copy a* to b*  
a1 copied TO b1  
a2 copied TO b2  
a3 copied TO b3  
-
```

The asterisks in the command are the wild card characters. In this instance, the command copies all files in the default directory that start with the character "A" to new files starting with the character "B." If files other than A1, A2, and A3 also begin with the character "A", this command will copy them also.

When you copy files, you can specify wild card characters (as in the previous example), lists of file names (as in the example before that), or a combination of both. However, some of the possible combinations are invalid. When copying files, remember the following rules:

- If you specify multiple input pathnames and a single output pathname, file concatenation takes place. This is described later in this chapter.
- If you specify multiple output pathnames, you must specify the same number of input pathnames as output pathnames. Specifying more input pathnames than output pathnames results in an error message. For example, the command:

```
-copy a,b,c to d,e (invalid)
```

returns an error message. The command:

```
-copy a,b to c,d,e (invalid)
```

also returns an error message. Refer to the "Inpath-List and Outpath-List" section of Chapter 2 for more information.

HUMAN INTERFACE EXAMPLES

HOW TO DISPLAY THE CONTENTS OF FILES

When you perform a number of file manipulations during a single session, it is occasionally advisable to display a file's contents at the terminal before proceeding further. Assume you wish to display the contents of a file named ALPHA that is contained in your default directory. Simply enter the command:

```
-copy alpha
aaaaa
aaaaa

alpha copied TO :CO:
-
```

This COPY command example uses the default preposition (TO) and default output file (:CO:), which means that the command copies the output to the console screen.

You can halt the scrolling of a displayed list to examine the data more closely. Press the following CTRL keys to control scrolling of the output:

CTRL/s	Stops the data from scrolling off the screen until you press a CNTRL/q.
CTRL/q	Resumes scrolling of listed data until the end-of-file is reached or you enter a CTRL/c.
CTRL/c	Cancels listing the data and returns control to the Human Interface, which prompts for a new command.

HOW TO REPLACE EXISTING FILES

There may be occasions when you wish to update the contents of an existing file. One way to do this is to create a new file and then replace the contents of the old file with the new data. Although you can use the RENAME command to perform this operation, this section shows how to replace the contents of a file with the COPY command's OVER preposition.

Assume the following conditions:

- You have a file named ALPHA that is accessed under that name by a number of different programs. ALPHA has outmoded data.
- Since you cannot change the name without also modifying the programs that access ALPHA, you must retain the name but update the outmoded file contents.

HUMAN INTERFACE EXAMPLES

Enter the following command sequence:

```
-copy :ci: to temp  
nu nu nu nu  
nu nu nu nu  
(CTRL/z)
```

```
:ci: copied TO temp  
-copy temp over alpha  
temp copied OVER alpha  
-copy alpha  
nu nu nu nu  
nu nu nu nu
```

```
alpha copied TO :CO:  
-
```

The last COPY ALPHA command lists the file at the terminal to show that the old file contents have been successfully replaced.

You could have used the TO preposition in the COPY command to write TEMP over ALPHA; but since the Human Interface always expects a new output file when the TO preposition is used, this would have caused unnecessary keystrokes, as shown in the following:

```
-copy temp to alpha  
alpha, already exists, OVERWRITE? y  
temp copied TO alpha  
-
```

Note that you now have two copies of the same new data; one in the TEMP file and one in the ALPHA file. If you had used the OVER preposition in a RENAME command instead of the COPY command, file TEMP would have been deleted automatically when RENAME was executed. However, if you did not want two existing copies of the same data, you could update the existing file directly from the keyboard. Enter the following command:

```
-copy :ci: over alpha  
newnewnew  
(CTRL/z)  
:ci: copied OVER alpha  
-
```

HOW TO CONCATENATE FILES

Concatenation is the process of combining a number of files by appending them in sequence into a single file. You can use the COPY command in several ways to concatenate files:

- by specifying the AFTER preposition in the command line
- by specifying multiple input pathnames and a single output pathname

HUMAN INTERFACE EXAMPLES

- by using a combination of both techniques

Assume you have four existing files named A, B, C, D respectively, and want to append the contents of B, C, and D to the end of file A. Although you could specify the TO preposition in the COPY command line, the TO preposition would force you to enter extra keystrokes because your listed output file (A) already exists. It would also force you to delete the previous contents of A, which is not always desirable. Therefore, use the AFTER preposition, as follows:

```
-copy b,c,d after a  
b copied AFTER a  
c copied AFTER a  
d copied AFTER a  
-
```

Now, assume you wish to concatenate all four files into a new file called ALL. You can still use the AFTER preposition, or you can use the TO parameter, as follows:

```
-copy a,b,c,d to all  
a copied TO all  
b copied AFTER all  
c copied AFTER all  
d copied AFTER all  
-
```

In this example, file A is copied to ALL and the remaining input files are automatically appended to the end of ALL.

You can save keystrokes when listing a series of files on the screen by using this automatic concatenation in a single command line. Assume you wish to list files named ALPHA, BETA, and GAMMA. Enter the following command, using the default TO preposition and default output file (:CO:):

```
-copy alpha,beta,gamma  
aaaaa  
aaaaa  
alpha copied TO :CO:  
bbbbbb  
bbbbbb  
beta copied AFTER :CO:  
gggggg  
gggggg  
gamma copied AFTER :CO:  
-
```

When data sequence and/or data format are important in a concatenated file, remember that all copy operations are performed in the sequence you specify in the command line.

Assume you have formatted data in a group of files named A, B, C, D, and E, and you wish to concatenate their contents into a new file named SQUARE in that sequence. However, if you list the input files on the command line in a haphazard sequence, as follows:

```
-copy b,a,d,c,e to square
```

HUMAN INTERFACE EXAMPLES

the format of the total data block is destroyed, as can be seen in the following incorrect and correct versions of the listed output. Although the data block of Latin words shown in the left-hand example seems correct when read horizontally, the intent and meaning of the vertical columns has been lost. The right-hand example shows the corrected file sequence:

<u>b,a,d,c,e</u> <u>sequence</u>	<u>a,b,c,d,e</u> <u>sequence</u>
A R E P O	S A T O R
S A T O R	A R E P O
O P E R A	T E N E T
T E N E T	O P E R A
R O T A S	R O T A S

In the right-hand example, the Latin "magic square" now reads the same both horizontally and vertically, which was the intended operation.

HOW TO DELETE FILES

It is vital to good file housekeeping that you routinely delete obsolete or unused files and empty directories. (Deleting unused directories is described later in this chapter.) In addition to the obvious benefit of recovering unused secondary storage, deleting your obsolete files reduces confusion and file manipulation errors.

Assume that you want to delete files ALPHA and BETA from the system. Enter the following command:

```
-delete alpha,beta  
alpha, deleted  
beta, deleted  
-
```

Now, assume that you entered the following command line and received the following error message:

```
-delete ay,bee,key  
ay, deleted  
bee, deleted  
key, does not exist  
-
```

The error message for the KEY file tells you one of three things:

- There is a syntax error in the spelling of the KEY file.
- The file does not exist.
- The file exists in a directory other than the one you are currently accessing (see the directory examples later in this chapter).

HOW TO USE DIRECTORIES

A directory is a kind of file under which you assign and maintain other files or directories. It is distinguished from a data file by a directory heading that is automatically created when you create a new directory. Under that heading, the directory maintains a formatted list of its containing files and directories. This heading is updated whenever you assign new files to the directory. Directories provide you with a convenient and efficient technique for organizing large numbers of files into logical groupings. Creating your own directories aids you in two ways:

- It allows you to organize your files into logical groupings. This eases the task of maintaining large numbers of files on the system.
- It reduces the possibility of accidental destruction of files, either by yourself or other system users.

A directory contains a list of all files assigned under its name, which you can display by using the DIR command (described later). Optional DIR command parameters also allow you to access and display other pertinent information about each file, such as file size and other file attributes.

Previous command examples in this chapter, when creating and accessing files, have used the default directory configured for your user ID. The following examples show you how to create and use your own directories for easier file management.

HOW TO CREATE A NEW DIRECTORY

Whenever you wish to group a series of files under a single topical structure, you normally create a new directory in which to assign them before creating the files themselves. (You can also move existing files under a new directory name by using the RENAME command, as described later.)

You create new directories by using the CREATEDIR command to specify a list of directory names for the new directories. You will find it easier to keep track of both your directories and files if you use directory names that give some hint of a directory's topical structure.

Assume you wish to create two directories named MYTEST and NUTEST under which you will assign several practice files. Enter the following command:

```
-createdir MYTEST,NUTEST
MYTEST, directory created
NUTEST, directory created
-
```

HUMAN INTERFACE EXAMPLES

This example specified the directory pathnames as uppercase characters. It is suggested that you also capitalize all directory pathnames in a CREATEDIR command and use lowercase characters for data pathnames when you create new files with the COPY command. This practice is recommended because, when you subsequently list a directory by using the DIR command (described later), it will be much easier for you to distinguish between data file names and directory names.

Once you create directories and data files, you can enter their pathnames in either lowercase or uppercase characters in subsequent commands; the Human Interface commands make no distinction in interpretation.

HOW TO REFER TO A DIRECTORY

After you create a new directory, all named files or directories that you assign to that directory will have a hierarchical relationship to this "parent" directory. This relationship to the parent is called a path. When you wish to access any file or other directory assigned to the parent, you must specifically identify the path in the form of a pathname in your command.

For example, assume your default directory has a directory named NUTEST under which you have another directory named SAMP. SAMP, in turn, has a data file named TEST. NUTEST is then the parent directory for the SAMP directory and SAMP, in turn, is the parent for the TEST data file. In a command, the pathname for the SAMP directory would be NUTEST/SAMP, where the slash characters separate the individual hierarchical components of the pathname. The pathname for the TEST data file would be NUTEST/SAMP/TEST.

If the files are contained in your default directory, you can refer to them without specifying a logical name as a prefix. When you enter the pathname:

```
NUTEST/SAMP/TEST
```

the Human Interface automatically appends the prefix :\$: to the beginning. However, if the files are contained in a directory other than your default directory, you must enter the complete pathname for the file. For example, if the files reside on a device whose logical name is :AD3:, you must include this logical name as the prefix portion of the pathname, as follows:

```
:AD3:NUTEST/SAMP/TEST
```

If you omit the :AD3: portion, the Human Interface assumes the files reside in the default directory.

HUMAN INTERFACE EXAMPLES

HOW TO ADD NEW ENTRIES TO A DIRECTORY

Previous data file examples in this chapter used the default directory (as configured for your system) for all file creation and access. Consequently, each example that created a new file or accessed an existing file specified only the last component of the file's pathname; it did not need to specify a logical name or intermediate pathname components. However, whenever you wish to create a new data file to be assigned to a specific directory, you must precede the filename with the directory name and separate the two names with a slash (/) in the COPY command, as described in the previous subsection. You might also need to specify a logical name, if the directories do not reside in your default directory.

For example, assume you wish to create files named SAMP1 and SAMP2 and assign them to the MYTEST directory (MYTEST resides in your default directory). Enter the following commands:

```
-copy :ci: to mytest/samp1  
aaaaa  
(CTRL/z)  
  
:ci: copied TO mytest/samp1  
-copy :ci: to mytest/samp2  
bbbbbb  
(CTRL/z)  
  
:ci: copied TO mytest/samp2  
-
```

Remember that once you have added files to a specific directory, every subsequent operation involving those files must specify a preceding directory name and the slash separator (unless you change your default directory, as described in a later section). For example, assume you want to delete files SAMP1 and SAMP2 from the MYTEST directory. You might enter the following command:

```
-delete mytest/samp1,samp2  
mytest/samp1, deleted  
samp2, does not exist  
-
```

The Human Interface issues the "does not exist" message for SAMP2 because it looked for the file in your default directory instead of the MYTEST directory. The correct command line entry should have been:

```
-delete mytest/samp1,mytest/samp2
```

so that the Human Interface would search the correct directory for each listed file.

HUMAN INTERFACE EXAMPLES

HOW TO CREATE A DIRECTORY WITHIN A DIRECTORY

In the same manner that you create new directories in your default directory, you can also create new directories in other directories, therefore expanding the file hierarchy.

For example, assume you have data files ALPHA, BETA, and GAMMA assigned to the MYTEST directory and now wish to add a new directory file named URTEST to the directory. Enter a CREATEDIR command, as follows:

```
-createdir mytest/URTEST  
mytest/URTEST, directory created  
-
```

Now, assume you wish to create a new data file named NOMOR and assign it to the URTEST directory. Enter the following COPY command:

```
-copy :ci: to mytest/urtest/nomor  
nononon  
nononon  
(CONTROL/z)  
  
:ci: copied TO mytest/urtest/nomor  
-
```

The "MYTEST/URTEST" sequence is the path from your default directory to the URTEST directory, and the "MYTEST/URTEST/NOMOR" sequence is the path from your default directory to the NOMOR file. When you use file-handling commands, you must always specify a path to the file, either a path from your default directory to the file, or a path from some other known point (such as from the root directory for another device). For example, assume you have another data file in URTEST named SUMOR and wish to list both NOMOR and SUMOR on the console screen. Enter the following command and specify the pathname for each file:

```
-copy mytest/urtest/nomor,mytest/urtest/sumor  
nononon  
nononon  
mytest/urtest/nomor copied TO :CO:  
sumsumsum  
sumsumsum  
mytest/urtest/sumor copied TO :CO:  
-
```

If the directory MYTEST resides on a device other than your default device (for example, :F6:), you would specify the previous command as follows:

```
-copy :f6:mytest/urtest/nomor,:f6:mytest/urtest/sumor  
nononon  
nononon  
:f6:mytest/urtest/nomor copied TO :CO:  
sumsumsum  
sumsumsum  
:f6:mytest/urtest/sumor copied TO :CO:  
-
```

HUMAN INTERFACE EXAMPLES

You can also specify file operations involving two or more different directories, and these directories need not be on the same path. Assume you wish to list the ALPHA file from MYEST and a file named DIFF on a directory path ONE/MOR. Enter the following command:

```
-copy mytest/alpha,one/mor/diff  
aaaaa  
aaaaa  
mytest/alpha copied TO :CO:  
yyyyy  
yyyyy  
one/more/diff copied TO :CO:  
-
```

HOW TO LIST DIRECTORIES

Previous examples have shown you how to list the contents of data files by specifying a directory pathname in a COPY command. However, you should not use the COPY command to list the contents of directories, because COPY lists the directory as though it were a data file. For example, if you enter the COPY command to list the MYTEST directory on the screen, you obtain:

```
-copy mytest  
alphabetagammaurtest copied to :CO:  
-
```

The resulting output is almost unreadable. Instead, use the DIR command to list the directory's catalog of files as follows:

```
-dir mytest  
  
01 JAN 78 00:00:00  
  DIRECTORY OF MYTEST ON VOLUME disk2  
alpha          beta          gamma  
URTEST
```

This example used the DIR command's default TO preposition and FAST format for the listing. You could have sent the directory listing to another output file and specified either the OVER preposition, to write the listing over the file's previous contents, or the AFTER preposition, to append the directory listing to other data. If you want to list more information about each file, specify the EXTENDED parameter. See the DIR description in Chapter 3 for examples of the available listing formats.

HUMAN INTERFACE EXAMPLES

HOW TO MOVE FILES BETWEEN DIRECTORIES

There may be situations when you wish to reorganize a large group of existing files under new headings (directories). You can move files from one directory to another by using the RENAME command. For example, assume you wish to move files ALPHA, BETA, and GAMMA from your default directory to the existing directory MYTEST, and file DELTA from your default directory to an existing directory named NUTEST. Enter the following command line, using the QUERY parameter (optional):

```
-rename alpha,beta,gamma,delta to MYTEST/alpha,MYTEST/beta, &  
**MYTEST/gamma,NUTEST/delta query  
alpha, rename TO MYTEST/alpha? y  
alpha renamed TO MYTEST/alpha  
beta, rename TO MYTEST/beta? y  
beta renamed TO MYTEST/beta  
gamma, rename TO MYTEST/gamma? y  
gamma renamed TO MYTEST/gamma  
delta, rename TO NUTEST/delta? y  
delta renamed TO NUTEST/delta  
-
```

Assume you later decide to move file ALPHA back to your default directory. You need not specify the default directory in the new pathname for ALPHA. Enter the following command:

```
-rename mytest/alpha to alpha  
mytest/alpha renamed TO alpha  
-
```

Any subsequent operations involving file ALPHA would only require the file name. For example:

```
-copy alpha  
aaaaa  
aaaaa  
alpha copied TO :CO:  
-
```

HOW TO DELETE A DIRECTORY

You delete unused directories from secondary storage by using the DELETE command. However, the Human Interface protects you from accidentally destroying valuable files by refusing to delete a directory that contains one or more files. For example, assume you wish to delete directory MYTEST and do not realize it contains a data file named ALPHA and a directory named DED that itself contains a data file named LIV. You enter the following command:

```
-delete mytest  
mytest, directory not empty  
-
```

HUMAN INTERFACE EXAMPLES

At this point you should list the MYTEST directory by using the DIR command to determine the contents of MYTEST, as follows:

```
-dir mytest  
  
01 JAN 78 00:00:00  
  DIRECTORY OF MYTEST  ON VOLUME disk2  
alpha                DED  
-
```

You now have two options. You can use the RENAME command to move any files to be saved to a different directory on the same volume, or you can use the DELETE command to delete the entire contents of MYTEST before deleting the directory.

Assume you wish to move ALPHA to the NUTEST directory and delete the rest of the directory's contents so that MYTEST itself can be deleted. Enter the following commands:

```
-rename mytest/alpha to nutest/alpha  
mytest/alpha renamed TO nutest/alpha  
-delete mytest/ded/liv,mytest/ded,mytest  
mytest/ded/liv deleted  
mytest/ded deleted  
mytest deleted  
-
```

The RENAME command automatically deleted the MYTEST/ALPHA pathname from the MYTEST directory. Note how the pathname sequence in the DELETE command travelled upward through the hierarchical structure, with the MYTEST directory being the last item to be deleted.

HOW TO CHANGE YOUR DEFAULT DIRECTORY

Suppose your default directory contains a directory called MYTEST which contains another directory called URTEST which in turn contains several data files called MOR, SUMOR, STILMOR, and NOMOR. If you plan to manipulate these data files extensively, your Human Interface commands can become very cumbersome, due to the length of the pathnames involved. For example, suppose you wish to copy the data files to files called ALPHA, BETA, DELTA, and GAMMA in the same directory. The command to do this is:

```
-copy mytest/urtest/mor, mytest/urtest/sumor, mytest/urtest/stilmor, &  
**mytest/urtest/nomor to mytest/urtest/alpha, mytest/urtest/beta, &  
**mytest/urtest/delta, mytest/urtest/gamma
```

If there are more levels in the directory structure, your commands can become even longer.

HUMAN INTERFACE EXAMPLES

To eliminate some of these long pathnames, you can use the ATTACHFILE command to change your default directory to be a directory closer to the level of the files with which you are working. To make the previous command shorter, you could change your default directory to the URTEST directory, as follows:

```
-attachfile mytest/urtest as :$:  
mytest/urtest attached AS :$:  
-
```

Now, when you make references to files without specifying the entire pathname, the Human Interface assumes that they reside in the URTEST directory, not your previous default directory. Therefore, to perform the same operation as in the previous COPY command, you could now enter the following command:

```
-copy mor, sumor, stilmor, nomor to alpha, beta, delta, gamma
```

You can use the ATTACHFILE command to change your default directory to any directory that you wish, so that you can manipulate the files in that directory more easily. To return to your original default directory, enter the following command:

```
-attachfile
```

This command uses the default parameters and has the same effect as "ATTACHFILE :HOME: AS :\$:". The :HOME: logical name represents your original default directory; therefore the command returns :\$: to its original value.

HOW TO RENAME FILES AND DIRECTORIES

The most direct method to save the contents of a file or directory but change its pathname is to use the the RENAME command. To make the process easier to follow, this section discusses the renaming of files and directories separately.

HOW TO RENAME FILES

Assume you wish to change the name of file ALPHA to a new name of OMEGA, where OMEGA does not already exist. Enter the following command:

```
-rename alpha to omega  
alpha renamed TO omega  
-
```

The ALPHA pathname is automatically deleted from the system when the RENAME command is executed. You can also rename lists of files to new pathnames. In this case, it is useful to include the QUERY parameter in your command line to make certain that your old pathnames and new pathnames are matched up in the way you intend.

HUMAN INTERFACE EXAMPLES

Assume you wish to rename files ALPHA, BETA, and GAMMA to TOM, DICK, and HARRY respectively. Enter the following command sequence:

```
-rename alpha,beta,gamma to tom,dick,harry  
alpha renamed TO tom  
beta renamed TO dick  
gamma renamed TO harry  
-
```

Remember that when using the RENAME command, you must always have a one-for-one match of pathnames between the new list and the old file list. For example, more old pathnames than new pathnames would cause the following exchange at the terminal:

```
-rename alpha,beta to tom  
alpha renamed TO tom  
TO or OVER preposition expected  
-
```

Similarly, specifying fewer old pathnames than new pathnames would cause the following exchange:

```
-rename alpha to beta,tom  
008B: E$UNMATCHED_LISTS  
-
```

So far, these RENAME examples have used the TO parameter to give new names to existing files. However, you can also use the OVER preposition with RENAME. The primary purpose of OVER is to move data from one named file over the data in another existing file. This matches the action of the OVER preposition in the COPY command with one important distinction: RENAME automatically deletes the input file when the command is executed.

Exercise a little care here! It's easy to get into semantic confusion when using the OVER preposition in a RENAME command. Just remember a few simple rules:

- Use the pathname of the data to be moved to a different but existing pathname as the input parameter; that is, on the left-hand side of the OVER preposition. This pathname will be deleted when the command is executed.
- Use the pathname that receives the input data as the output parameter; that is, on the right-hand side of the OVER preposition. The previous contents of this file will be replaced when the command is executed.

For example, assume you have a file named ABLE whose contents consist of the data line aaaaa, and another file named BAKER whose contents consist of the data line bbbbb. You wish to rename ABLE with the name BAKER. Enter the following command:

```
-rename able over baker  
able renamed OVER baker  
-
```

HUMAN INTERFACE EXAMPLES

Now display the contents of the file previously named ABLE but now named BAKER:

```
-copy baker  
aaaaa
```

```
baker copied TO :CO:  
-
```

The previous contents of BAKER have been deleted, and pathname ABLE has been deleted from its directory. You can also use the TO preposition to rename files with other existing pathnames. Using TO might be slightly less confusing but you must enter extra keystrokes. For example, assume you wish to rename ALPHA and BETA with the existing file names GAMMA and DELTA. Enter the following command:

```
-rename alpha,beta to gamma,delta  
gamma, already exists, OVERWRITE? y  
alpha renamed TO gamma  
delta, already exists, OVERWRITE? y  
beta renamed TO DELTA  
-
```

HOW TO RENAME DIRECTORIES

A directory can be renamed to new pathname on the same volume (but not to an existing pathname). Assume you have a directory whose pathname is ALPHA/BETA and you wish to rename it with a new pathname of AY/BEE. Enter the following command:

```
-rename alpha/beta to AY/BEE  
alpha/beta renamed TO AY/BEE  
-dir alpha/beta  
alpha/beta, does not exist  
-
```

Be careful when renaming directories! The last message explains the consequences of renaming a directory to a new pathname. Once you rename a directory, all files listed under that directory will also have their pathnames changed. If your system has other programs that use data files that are listed under the old directory name, those programs will never find the files. In such a case, you must either rename the directories to their original names or modify the programs.

In summary, the distinctions between using the RENAME and COPY commands are as follows:

- When you use COPY to move the contents of an existing file TO a new file or OVER an existing file, the input file still exists.
- When you use RENAME to move the contents of an input file TO a named new file or OVER an existing file, the input pathname is automatically released for new uses.

HOW TO MOVE FILES ACROSS VOLUME BOUNDARIES

You can use all Human Interface file-handling commands except RENAME to manipulate files across volume boundaries. That is, you can copy files or directories from one diskette or disk platter to another one mounted on a different drive. The restriction against using RENAME across volume boundaries is intended for the protection of files against accidental deletion.

You access a different volume by entering the logical name for the device (the drive on which the volume is mounted) as the first item in the pathname. For example, assume you have a volume mounted on a drive whose logical name is :fl:. Further assume you wish to list the root directory for that volume to see what directories and data files you have on the volume. Enter the following command:

```
-dir :fl:
```

```
01 JAN 81 00:00:00
  DIRECTORY OF :fl: ON VOLUME disk2
able          baker          chuck          OMNI          samp
BUS           nusamp         STATS
```

-

Assume you wish to copy file ABLE from this volume mounted on :fl: to the MYTEST directory (which resides in your default directory). Enter the following command:

```
-copy :fl:/able to mytest/able
:fl:/able copied TO mytest/able
```

-

If you then wish to delete files ABLE and BAKER from the :fl: volume, simply enter the command:

```
-delete :fl:/able,:fl:/baker
:fl:/able, deleted
:fl:/baker, deleted
```

-

Now, assume the following conditions:

- You have two data files on the :fl: volume with the pathnames STATS/SALES/FEB and STATS/SALES/MAR.
- You wish to merge both files to a new file with the pathname MYTEST/PEEK/SUBTOT on your system's default volume.

Enter the following command:

```
-copy :fl:/stats/sales/feb,:fl:/stats/sales to mytest/peek/subtot
:fl:/stats/sales/feb copied TO mytest/peek/subtot
:fl:/stats/sales/mar copied AFTER mytest/peek/subtot
```

-

Note that a volume prefix must be specified for each pathname in any command that crosses volume boundaries. A volume uses the prefix of the drive on which it is mounted.

HOW TO FORMAT A NEW VOLUME

Whenever you wish to use a new volume on a secondary storage device (such a diskette, disk platter, or bubble memory), you must format the volume before you can write any information in it. Assume you are going to mount a new diskette on a disk drive with the prefix :fl: that you have attached (with the ATTACHDEVICE command) as a named device.

Enter the following command:

```
-format :fl:
volume ( ) will be formatted as a NAMED volume
  granularity   = 128           sides       = 1
  interleave    = 5             density     = single
  files         = 50            disk size  = standard (8")
  extensionsize = 3
  volume size   = 250K
volume formatted
-
```

This formatting example exercised all the default options.

This example did not specify a volume name as parameter of FORMAT. A volume name is not required; however, for diskettes, a volume name gives you a method for identifying a volume in case the stick-on label on the diskette gets lost or destroyed. You need only mount the disk on a drive and enter a DIR command for that drive to get a directory listing that specifies the volume name.

The GRANULARITY, INTERLEAVE, EXTENSIONSIZE, and FILES parameters tell the FORMAT command how you want the physical space (for instance, disk surface space) on the volume allocated and accessed for maximum efficiency. The default parameters caused the NEWVOL example to be formatted with the following attributes:

- Since the device is attached as a named device, the NAMED parameter is the default with FORMAT. It specifies that you will be using the volume only to handle named files and directories. If you specified the PHYSICAL parameter, the entire volume would be treated as a single, large physical file. Once you define the volume as NAMED or PHYSICAL, you can only use it for that purpose.
- The GRANULARITY parameter specifies the minimum number of bytes to be allocated for each increment of file size on the volume. The default granularity is the granularity of the physical device. Once the volume granularity is defined, it is applied to every file you create on the volume.

HUMAN INTERFACE EXAMPLES

For example, assume the default volume granularity for your device is 1024 bytes. Each time you create a new file on the volume, the I/O System automatically allocates 1024 bytes of primary storage to that file, whether or not the file requires the full 1024 bytes. If the size of your file exceeds 1024 bytes, the I/O System will increment your file size by still another block of 1024 bytes, and so on, until the end-of-file is reached.

- The INTERLEAVE default specifies that you want an interleave factor of 5. The interleave factor defines the number of physical sectors that occur between sequential logical sectors. This value maximizes access speed for the files on a given volume, depending upon the intent of the volume and the device configuration of your system.

For example, an interleave value of 5 for a flexible disk system means that, for each file, the I/O System will read every fifth sector on the diskette, starting from an index of 1 (other hard disk systems may be different, depending on your hardware configuration). Therefore, the I/O System does not need to wait for the disk to make a complete revolution before it accesses the next sector; the next sector by an increment of 5 is ready to be accessed for read/write by the time the first accessed sector has been processed.

Note that the INTERLEAVE is the only optional parameter that is meaningful for volumes formatted for PHYSICAL files; the FILES, EXTENSIONSIZE, and GRANULARITY options are ignored in FORMAT commands that specify a PHYSICAL file format for the volume.

- The default FILES parameter specifies that you wish create a maximum of 50 user files on the volume. Although the actual number of files you can specify is 1 through 32,761, at a practical level, one of your determining factors will be the incremental file size you specify in the GRANULARITY parameter.

The default EXTENSIONSIZE parameter specifies that you wish to create three bytes of extension data for each file. The Human Interface requires that at least three bytes of extension data be available. Other system programs included in your system may require larger values.

DISKETTE SWITCHING PROCEDURES

If your system is configured with the iSBC 204 flexible disk controller and you are using single-density diskettes to perform file management functions, a special procedure is required to switch the diskettes. Perform the following steps:

1. Remove the old diskette and mount the new one into the drive.

HUMAN INTERFACE EXAMPLES

2. Enter a DIR command for the root directory of the new diskette to force physical access. The root directory "name" is actually the prefix (logical name) for the drive on which the diskette is mounted. For example:

-dir :fl:

The following exception message will be displayed:

E\$IO

-

3. Ignore the error message and begin entering Human Interface commands that access the volume.

CHAPTER 5. PATCHING UTILITY

The iAPX 86, 88 Patching Utility is a utility that runs on both iRMX 86 application systems and Series III Microcomputer Development Systems. It modifies combine-type attributes of object modules, permitting them to be written over (or repaired) with replacement modules. This provides you with a method of modifying relocatable object modules with software updates or repair code. The process requires that the replacement code first be generated with the ASM86 Assembler.

The Patching Utility also enables you to list the translator header records of the modules in a library. This allows you to see which patches have been installed in a module or a library.

TYPES OF PATCHES

You can update a module to a newer version or add repair code in two ways:

- As a patch that generates a jump instruction to the replacement code and appends the replacement code to the end of the original module.
- As an in-place patch that directly overlays the replacement code on that of the original module.

An example of each technique is provided later in this chapter.

TYPES OF REPLACEMENT CODE

The replacement code itself may be supplied by Intel in either of two forms:

- As an Intel-supplied object file, on diskette. In this case, all of the coding and assembly has been done. You need only invoke the Patching Utility to effect the replacement.
- As an Intel-supplied source code listing with instructions for inserting the replacement code. In this case, much of the preliminary work has been done; you need little or no knowledge of ASM86 Macro Assembly Language to generate the replacement object module.

You can also use the Patching Utility to modify your application modules with replacement modules that you create. In this case, a working knowledge of ASM86 Macro Assembly Language is required.

PATCHING UTILITY

VERSIONS OF THE PATCHING UTILITY

There are two versions of the Patching Utility. One version runs on an iRMX 86 application system and can be invoked with a Human Interface command. The other version run on a Series III Microcomputer Development System and can be invoked using the Series III RUN command. Both versions are contained on the Utilities release diskette and have the following file names:

<u>Version</u>	<u>File Name</u>
iRMX 86	PTCH86.R86
Series III	PTCH86.86

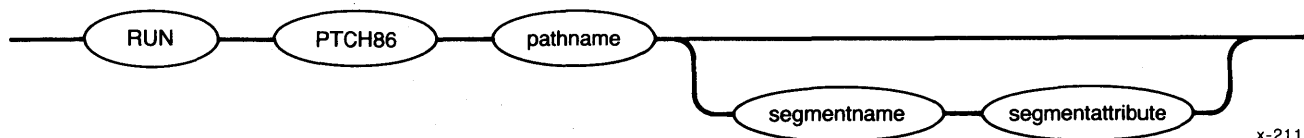
If you intend to use the Series III version, copy PTCH86.86 from the Utilities release diskette to a file on one of your Series III disks. If your release diskette is in iRMX 86 format, this involves using the iRMX 86 DOWNCOPY command (refer to Chapter 3) or the Files Utility (refer to Chapter 6). Otherwise, use the COPY command available with the Series III.

If you intend to use the iRMX 86 version, copy PTCH86.R86 to one of your iRMX 86 secondary storage devices and change its name to PTCH86. If your release diskette is in ISIS-II format, this involves using the Human Interface UPCOPY command (refer to Chapter 3) or the Files Utility (refer to Chapter 6). Otherwise, use the COPY command available with the Human Interface.

INVOKING THE PATCHING UTILITY

Before invoking the Patching Utility, ensure that the file containing it resides in the proper place. If you are using the Series III version, ensure that the PTCH86.86 resides on drive 0 of your development system. If you are using the iRMX 86 version, ensure that PTCH86 resides in your default directory or in one of the directories that the Operating System automatically searches (usually :PROG: and :SYSTEM:). Then you can invoke the Patching Utility by entering one of the following commands:

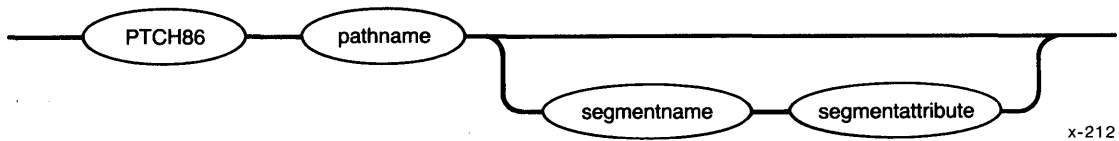
Series III Invocation



x-211

PATCHING UTILITY

iRMX 86 Invocation



where:

pathname Pathname of the file containing an iAPX 86, 88 object module produced by PL/M-86, ASM86, or LINK86. If you are listing the translator header records, the object module can be one produced by LOC86 or LIB86.

segmentname Name of the segment whose combine-type attribute is to be modified. The name must be a valid segment name (usually CODE). If you omit this parameter (and the segmentattribute parameter), the utility does not modify the attributes of any segment. Instead, it displays the translator header records contained in the module.

segmentattribute Combine-type attribute to be given to the named segment. You must specify one of the following values for this attribute:

- COMMON Allows patch code to be overlaid on the segment.
- PUBLIC Returns the segment to the combination mode normally given by the PL/M-86 compiler.

Refer to the ASM86 LANGUAGE REFERENCE MANUAL for a more detailed explanation of combine-type attributes.

In response, the Patching Utility displays the following header message:

```
<operating system> iAPX 86, 88 OBJECT PATCHING UTILITY, Vx.y
```

where:

<operating system> Name of the operating system, either iRMX 86 or SERIES III.

x.y Version number of the Patching Utility.

PATCHING UTILITY

If you exclude the optional `segmentname` and `segmentattribute` parameters, the utility next displays the translator header records contained in the object file. These header records serve to identify the patches that have been made to a module (described later in this chapter). This display allows you to determine the update status of the file.

If you specify the `segmentname` and `segmentattribute` parameters, the utility omits the display of the translator header records. Instead, it changes the combine-type attribute as specified and displays the following message:

ATTRIBUTE MODIFIED

Any other message indicates an error condition.

ERROR MESSAGES

When the Patching Utility encounters an error condition during a module repair session, it displays one of the following error messages:

- ERROR nnn

When using the Patching Utility on a Series III development system, the operating system returned an error message. Refer to the INTELLEC SERIES III MICROCOMPUTER DEVELOPMENT SYSTEM CONSOLE OPERATING INSTRUCTIONS for a description of numbered messages of this form.

- <exception code value> : <exception code mnemonic>

When using the Patching Utility on an iRMX 86 application system, the operating system returned an exceptional condition code. Refer to Appendix A of this manual for summary of these condition codes. Refer to the appropriate iRMX 86 manual for detailed descriptions of iRMX 86 exception codes.

- INVALID MODULE TYPE

The file specified is not suitable for the segment attribute modification and subsequent patching.

- INVALID RECORD TYPE

The object file contains an invalid record type for the object module format. Perhaps you entered the wrong file name or specified a file that contains code other than object code.

PATCHING UTILITY

- INVALID SYNTAX

The command line contains an error that was caused by a missing file name or a missing or misspelled keyword.

- SEGMENT NOT FOUND

The desired record was not found before the end of the module.

PATCHING PROCEDURES

Repair modules that you insert into existing modules must be generated with the ASM86 Assembler. To patch an independent object module containing errors (patching library modules is described later in this chapter), you must first invoke the Patching Utility to modify the combine-type attribute in the desired module segment to COMMON. This step allows you to use LINK86 to overlay the repair module on the segment to be patched. After linking with the repair module, you then use the Patching Utility to restore the PUBLIC attribute to the segment. The following example illustrates the steps for repairing independent object module files:

1. Enter the PTCH86 command to set the CODE segment combine-type attribute to COMMON. For example, on a Series III development system, enter:

```
RUN PTCH86 badmodule CODE COMMON
```

2. Enter the LINK86 command to overlay the repair object module on the original version, for example:

```
RUN LINK86 badmodule, repairmodule TO newmodule
```

3. Enter the PTCH86 command to restore the CODE segment to PUBLIC, for example:

```
RUN PTCH86 newmodule CODE PUBLIC
```

Typical examples of jump instruction overlays, in-place patch overlays, library module patching, and listing module header records are given in the following sections.

JUMP INSTRUCTION PATCH

In the following example, the module generates a patch that overlays a three-byte jump instruction on offset 0100H through 0102H of the original module. The jump transfers control to repair code at offset 0500H. The repair code is appended to the end of the module.

PATCHING UTILITY

EXAMPLE:

```
NAME    REPAIR_V00001      ; Identifying module name.

CODE    SEGMENT          WORD COMMON    'CODE'
CGROUP  GROUP           CODE
        ASSUME          CS : CGROUP

        ORG            0100H      ; Offset of area in original
                                   ; module to be patched.

        JMP            REPAIRCODE

RETURN  LABEL          NEAR      ; Return here from repair area.

        ORG            0500H      ; Offset of end of original
                                   ; module.

REPAIRCODE:
;
;   (Repair goes here)
;
        JMP            RETURN    ; Return control to original
                                   ; module.

CODE    ENDS
END
```

When making a jump instruction patch similar to the one listed previously, you must overlay the three-byte jump instruction on one of the following:

- A three-byte instruction
- A two-byte instruction and a one-byte instruction
- Three one-byte instructions

Otherwise, you must place NOP instructions after the JMP instruction and before the RETURN label so that the repair code returns to the start of the next instruction (not to the middle of a previous instruction).

IN-PLACE PATCH

The following example generates an in-place patch that directly overlays repair code on a module's previous code.

PATCHING UTILITY

EXAMPLE:

```
NAME      REPAIR_V00002      ; Module name identification.

CODE      SEGMENT          WORD COMMON      'CODE'
CGROUP    GROUP           CODE
          ASSUME          CS : CGROUP

          ORG             0200H      ; Offset of the original operand.

          ADD             AX, 3      ; Replaces the original value with a "3"
                                   ; (the new instruction must be the same
                                   ; size as the original instruction).

CODE      ENDS

END
```

PATCHING LIBRARY MODULES

To patch an object module that is located in a library, use one of two SUBMIT files supplied on your Utilities diskette: PATCH.CSD or PATCH.CMD. PATCH.CSD is designed to run on Series III development systems; PATCH.CMD is designed to run on iRMX 86 application systems. If you plan to use a Series III development system, you should copy PATCH.CSD to the same volume that contains the Patching Utility. If you plan to use an iRMX 86 system, you should copy PATCH.CMD to a directory that the Human Interface automatically searches. Each SUBMIT file expects the LINK86 and LIB86 utilities to be present.

When invoked, each SUBMIT file performs the following steps:

1. Invokes the LINK86 command to separate the module to be patched from the library and put it in a temporary file.
2. Invokes the PTCH86 command to set the CODE segment combine-type attribute to COMMON.
3. Invokes the LINK86 command to overlay the replacement object module on the original version.
4. Invokes the PTCH86 command to restore the CODE segment PUBLIC attribute.
5. Invokes the LIB86 command to replace the original module in the library with the updated version.
6. Deletes the temporary files when the replacement is completed.

To invoke either SUBMIT file, enter one of the following commands. Note that the parentheses enclosing the parameter string and the embedded commas are required; embedded blanks are optional:

PATCHING UTILITY

Series III command

SUBMIT PATCH(library, oldmodule, segment, newmodule)

iRMX 86 command

SUBMIT PATCH.COMD(library, oldmodule, segment, newmodule)

where:

library	Pathname of library containing the old module to be replaced.
oldmodule	Name of the module to be replaced.
segment	Name of the segment whose combine-type attribute is to be set to COMMON.
newmodule	Pathname of the file containing the replacement module code.

LISTING TRANSLATOR HEADER RECORDS

If you want the Patching Utility to list an object module's translator header records on the console screen, enter the PTCH86 command without specifying the segment name or segment attribute. The listed records allow you to identify the patches that have been made to the module. A typical PTCH86 command entry (from an iRMX 86 system) and resulting header record display is as follows:

```
-PTCH86 FILE.OBJ  
iRMX 86 IAPX 86, 88 OBJECT PATCHING UTILITY, Vx.y  
  ORIGINALMODULE  
  ORIGINALMODULE_REPAIR_V030-01  
  ORIGINALMODULE_REPAIR_V030-02
```

The "030" stands for version 3.0 of the software being patched, and "01" and "02" are the patch numbers of the Intel-supplied patches that have been made to the module. The Patching Utility can perform this listing operation on both object modules and libraries.

CHAPTER 6. FILES UTILITY SYSTEM

Because INTELLEC Microcomputer Development Systems do not recognize iRMX 86 diskette files, you cannot read, write, or format iRMX 86 diskettes directly from the ISIS-II operating system. However, you can perform these operations indirectly from the Development System by using the iRMX 86 Files Utility System. The iRMX 86 Files Utility System is an iRMX 86 application system that allows you to perform the following operations:

- Format an iRMX 86 diskette.
- Copy a file from an ISIS-II diskette to an iRMX 86 diskette.
- Copy a file from an iRMX 86 diskette to an ISIS-II diskette.
- Delete a file from an iRMX 86 diskette.
- Create a directory on an iRMX 86 diskette.
- Display, on the Development System terminal, the contents of a directory of an iRMX 86 diskette.

If you cannot use the startup system (described in the iRMX 86 INSTALLATION GUIDE) to format your first iRMX 86 secondary storage volumes and transfer necessary files (such as Human Interface commands) to these volumes, you can use the Files Utility for this purpose. The Files Utility System also gives you the ability to build and maintain secondary storage volumes for iRMX 86 application systems that do not include the Human Interface.

HARDWARE REQUIRED

The Files Utility System requires the following hardware:

- A Series III Microcomputer Development System having at least 64k bytes of memory and at least one disk drive (hard or flexible).
- A target system consisting of an iAPX 86, 88-based Single Board Computer, at least 192k bytes of memory, and at least one disk drive (hard or flexible).
- The iSBC 957B iAPX 86,88 Interface and Execution Package.

FILES UTILITY SYSTEM

STARTING THE FILES UTILITY

Before you can enter commands to the Files Utility, you must start it up. This involves connecting certain hardware modules and then entering appropriate commands at the Series III terminal.

After you have assembled your hardware, perform the following steps:

1. Place an ISIS-II system diskette containing the iSBC 957B software into drive 0 of your INTELLEC Microcomputer Development System and the Utilities release diskette into any other drive.
2. Load the ISIS-II system.
3. Enter the following ISIS-II command:

```
SUBMIT :fx:FILES (:fx:)
```

where:

fx Identifier of the diskette drive containing the
 Utility release diskette.

When you enter this command, the ISIS-II operating system reads and processes the commands contained on the FILES.CSD file. These commands instruct the iSBC 957B monitor to load the Files Utility System from a diskette on the INTELLEC system into RAM on the target system.

After the ISIS-II system finishes processing the commands in the SUBMIT file, the system prompts for another command. Respond by entering:

```
APXLOD
```

This command instructs the ISIS-II system to connect you to the monitor. The monitor then signals you that it is ready to accept your next command by displaying a period (.) on the screen of your Series III system. When the period appears, enter:

```
G
```

This causes the Disk Utility System to begin running. The screen of your INTELLEC system should display the heading:

```
iRMX 86 FILES UTILITY Vx.x '
```

The Files Utility signals that it is ready to accept your next command by displaying an asterisk (*) at the screen of the INTELLEC system.

FILES UTILITY SYSTEM

USING THE FILES UTILITY

The Files Utility provides 10 file management commands, as follows:

ATTACHDEV	DIR
BREAK	DOWNCOPY
CREATEDIR	FORMAT
DELETE	HELP
DETACH	UPCOPY

The commands are described in alphabetical sequence later in this chapter. However, before actually using the commands, you should understand the diskette handling procedures and how the Files Utility System handles errors.

CHANGING DISKETTES

When the Files Utility is running and you have already performed an operation on a particular diskette, you cannot simply remove that diskette from the drive and replace it with another. The Utility System is not aware of diskette changes and treats the second diskette as if it were the first, and thereby possibly writes over or destroys valuable information. To change diskettes in a drive, you must enter a DETACH command to logically detach the drive from the system, change diskettes, and then (with one exception) enter an ATTACHDEV command to again logically attach the device.

The one exception to this command entry sequence is the FORMAT command. As described later in this chapter, this command writes iRMX 86 formatting information on blank diskettes. Since the FORMAT command always expects a blank diskette and a detached drive, you can replace diskettes in a drive any number of times if you use only the FORMAT command before entering the ATTACHDEV command. The FORMAT command destroys the information, if any, previously contained on the diskette.

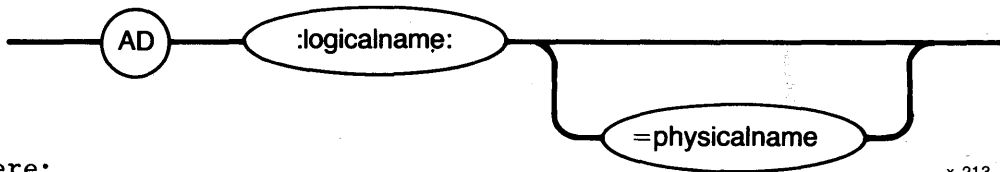
COMMANDS

This section provides descriptions of the Files Utility commands and their parameters in alphabetical sequence. Each command has a two-character abbreviation. You can use either the full name or its abbreviation when entering a command.

ATTACHDEV (AD)

This command attaches a physical device to the system and associates a logical name with the device. The command can also be used to display the current attachment of a logical name. The format is as follows:

FILES UTILITY SYSTEM



where:

x-213

:logical-name: A 1-to 12-character ASCII name, surrounded by colons.

=physical-name If used, there must be no spaces surrounding the equal sign. This specifies the physical device name as configured in the I/O System (see Table 3-2). If physical name is omitted, the current attachment is displayed by default; for example:

AD :FO: (command entry)
:FO: = FX0 (displayed output)

BREAK (BR)

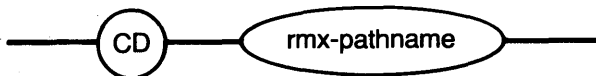
This command causes an exit from the Files Utility System to the monitor. The format is as follows:



x-214

CREATEDIR (CD)

This command creates an iRMX 86 directory file on the attached device. The format is as follows:



x-215

where:

rmx-pathname Pathname of the iRMX 86 directory file to be created.

FILES UTILITY SYSTEM

DELETE (DE)

This command removes the specified iRMX 86 file from the directory where it is listed. The format command is as follows:

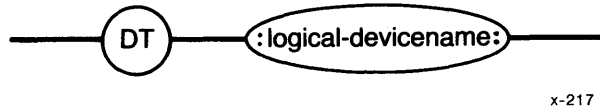


where:

rmx-pathname Pathname of the iRMX 86 file to be deleted.

DETACH (DT)

This command detaches a logical name from the system. The command is used for changing diskettes, prior to entering a FORMAT command, or to reconfigure a device to a different sector size. The format is as follows:

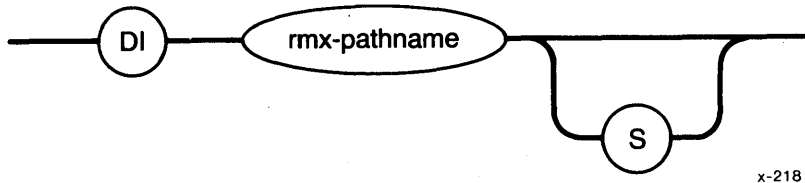


where:

:logical-name: The logical name you assigned to a physical device via an ATTACHDEV command.

DIR (DI)

This command lists an iRMX 86 directory file at the Development System console. The format is as follows:



where:

rmx-pathname Pathname of the iRMX 86 directory file to be listed.

FILES UTILITY SYSTEM

S Switch that causes a "long" or expanded display of directory file that includes: file type (a "DR" heading for a directory file, a "MP" heading for the bit map file, or a blank heading for a data file), number of blocks, and number of bytes in file. If S is not specified, a "fast" format will be displayed, consisting of file names only.

The directory file listing includes a line that lists the size of the directory. This line appears as:

<n> FILES

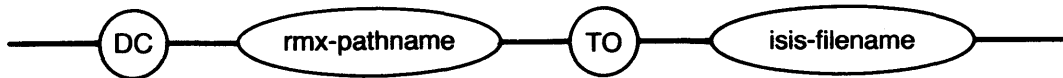
In this line, <n> specifies the number of entries currently present in the directory. If you specify the S parameter, this command also lists the following information about the directory:

<numblks> BLOCKS <numbytes> BYTES

In this line, <numblks> specifies the number of volume-granularity blocks allocated to files in the directory and <numbytes> specifies the number of bytes allocated to files in the directory.

DOWNCOPY (DC)

This command creates an ISIS-II file and copies the specified iRMX 86 file to it. If the ISIS-II file already exists, it is written over. The format is as follows:



x-219

where:

rmx-pathname Pathname of the iRMX 86 file to be copied.

isis-filename Name of the ISIS-II file to be created.

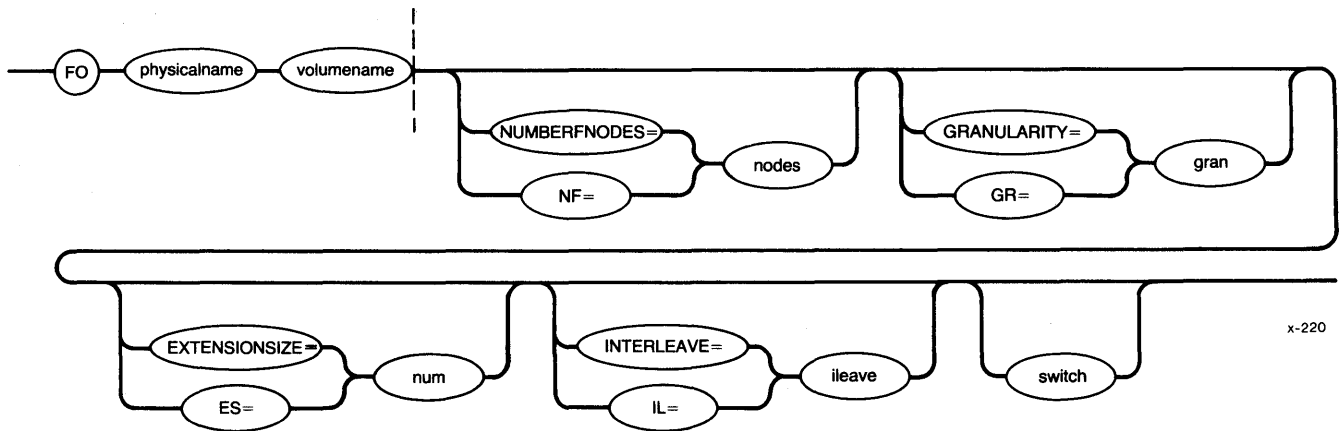
FORMAT (FO)

This command writes iRMX 86 formatting information on a secondary storage device. It performs the same kind of operations as the Human Interface FORMAT command described in Chapter 3. All information previously contained on the device is destroyed by the formatting operation. Each device must be formatted before it can be used by the iRMX 86 Operating System.

FILES UTILITY SYSTEM

The **FORMAT** command expects an unattached device. The device can either be unattached at system start up, or you can detach it by entering a **DETACH** command prior to entering the **FORMAT** command. Since the device remains unattached after **FORMAT** completes execution, you must attach the device by entering an **ATTACHDEV** command before entering any other Utility command except another **FORMAT** command. (See also the "Changing Diskettes" section in this chapter, and the **ATTACHDEV** and **DETACH** command descriptions.)

The **FORMAT** command contains parameters that are specified in the form "keyword=value". When entering parameters of this type, you must not place any spaces around the equal sign. Also, you can abbreviate each of these keywords as shown. The abbreviations and the format of this command are as follows:



where:

physicalname Physical device name for the drive, as configured in the I/O System, that denotes the iRMX 86 drive on which the diskette resides. Possible values are itemized in Table 3-2.

volumename A 1- to 10-character volume name that identifies the diskette. Decimal digits, uppercase and lowercase letters, and the following special characters can be used in the volume names:

! & , * ;
 " ' (+ /
 % .) : = ?

nodes The number of files (including internal system files) that can be created on this volume. If you omit this parameter, a default value of 56 is assumed.

FILES UTILITY SYSTEM

- gran** The granularity, in bytes, for this volume. The granularity is the number of bytes obtained during each diskette access. If you omit this parameter, the default volume granularity is the device granularity (the number of bytes in a physical sector). Specifying any value less than the device granularity causes the default to be used. Any non-multiple of device granularity (such as 128 or 512) is rounded upward to the next higher multiple of device granularity.
- num** Size, in bytes, of the extension data associated with each file. This data is used by A\$GET\$EXTENSION\$DATA and A\$SET\$EXTENSION\$DATA system calls (refer to the iRMX 86 BASIC I/O SYSTEM REFERENCE MANUAL). The Human Interface requires files it accesses to have three bytes of extension data. The range is 0 through 255 (decimal). If not specified, the default is three bytes.
- ileave** The interleave factor for the volume, or the number of physical sectors between logical sectors. You can specify any integer from 1 to 13 for this value. If you omit this parameter, a default value of 5 is assumed.
- switch** A switch that indicates the support option for this volume. One value can be entered for the switch:

NAMED The volume is created to contain named files. The ROOT directory is initialized.

If you omit this switch, the volume is created as a single physical file. In this case, FORMAT records the interleave information on the diskette but does not initialize any of the iRMX 86 file structures.

When it formats a named volume, the FORMAT command creates six internal system files. It names three of these files and lists their names in the root directory of the volume. The files are:

<u>file</u>	<u>description</u>
R?SPACEMAP	Volume free space map
R?FNODEMAP	Free fnodes map
R?BADBLOCKMAP	Bad blocks map

The command assumes that the user WORLD is the owner of these files. Refer to the iRMX 86 DISK VERIFICATION UTILITY REFERENCE MANUAL for more information about these files.

FILES UTILITY SYSTEM

HELP (HE)

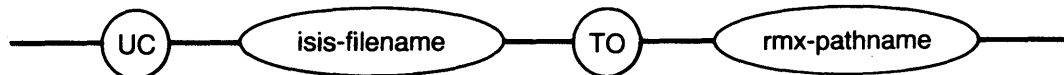
This command displays a list of the available Files Utility commands and their syntax on the console screen. The format is as follows:



x-222

UPCOPY (UC)

This command creates an iRMX 86 file and copies the specified ISIS-II file to it. If the iRMX 86 file already exists, it is written over. The format is as follows:



x-223

where:

isis-filename	Name of the ISIS-II file to be copied.
rmx-pathname	Pathname of the iRMX 86 file to be created.

ERROR MESSAGES

The Files Utility displays all error messages on the screen of the INTELLEC Microcomputer Development System. These messages can be in any of three forms. They are:

- UNRECOGNIZED COMMAND

The Files Utility does not recognize the spelling of your command. It prompts for another command.

- ISIS ERROR # <nnn>

The Files Utility actually uses the ISIS-II operating system to read and write diskettes attached to the INTELLEC Microcomputer Development System. If the ISIS-II system detects any errors, it returns an error code to the Files Utility. To interpret this error message, refer to the INTELLEC SERIES III MICROCOMPUTER DEVELOPMENT SYSTEM CONSOLE OPERATING INSTRUCTIONS. Fatal errors require you to restart the Files Utility System by using the FILES.CSD file, as described earlier in this chapter.

FILES UTILITY SYSTEM

- RMX EXCEPTION # <mmmm>

When reading or writing on drives attached to the target system, the Files Utility System uses the iRMX 86 Nucleus and the iRMX 86 I/O System. If either of these layers returns an exceptional condition code, the Files Utility displays the condition code in this format, where mmmm is in hexadecimal. For a brief explanation of such an error message, refer to Appendix A. After displaying this message, the Files Utility prompts for the next command.

APPENDIX A. CONDITION CODES SUMMARY

Table A-1 provides a list of the iRMX 86 condition codes that can occur during system operations. This table provides a minimum of information about each condition code. In most cases, the condition code must be considered in terms of the unique circumstances that caused the condition. Table A-1 is provided to guide you to the most appropriate manual. The appropriate iRMX 86 manuals have more detailed descriptions of the meanings. The appropriate manual is listed in the column marked "Manuals".

Table A-1. iRMX™ 86 Condition Codes

Hex. Value	Mnemonic	Manuals N B E L H	Meaning
0H	E\$OK	* * * * *	No exceptional conditions (normal)
Environmental Conditions			
1H	E\$TIME	* * * * *	A time limit (possibly a limit of zero time) expired without a task's request being satisfied.
2H	E\$MEM	* * * * *	Insufficient available memory to satisfy a task's request.
3H	E\$BUSY	*	Another task currently has access to data protected by a region.
4H	E\$LIMIT	* * * * *	A task attempted an operation which, if it had been successful, would have violated a Nucleus-enforced limit.
5H	E\$CONTEXT	* * * * *	A system call was issued out of proper context.
N Nucleus Reference Manual		L Loader Reference Manual	
B Basic I/O System Ref Manual		H Human Interface Reference Manual	
E Extended I/O Sys Ref Manual			

CONDITION CODES SUMMARY

Table A-1. iRMX™ 86 Condition Codes (continued)

Hex. Value	Mnemonic	Manuals N B E L H	Meaning
Environmental Conditions (continued)			
6H	E\$EXIST	* * * * *	A token parameter has a value which is not the token of an existing object.
7H	E\$STATE	*	A task attempted an operation which would have caused an impossible transition of a task's state.
8H	E\$NOT\$CON- FIGURED	* * * * *	This system call is not part of the present configuration.
9H	E\$INTER- RUPT\$SAT- URATION	*	An interrupt task has accumulated the maximum allowable amount of SIGNAL\$INTERRUPT requests.
0AH	E\$INTER- RUPT\$- OVERFLOW	*	An interrupt task has accumulated more than the maximum allowable amount of SIGNAL\$INTERRUPT requests.
20H	E\$FEXIST	* *	File already exists.
21H	E\$FNEXIST	* * * *	File does not exist.
22H	E\$DEVFD	* * *	Device and file driver are incompatible.
23H	E\$SUPPORT	* * * *	Combination of parameters not supported.
24H	E\$EMPTY\$- ENTRY	* *	The specified slot in a directory file is empty.
25H	E\$DIR\$END	* *	The specified slot is beyond the end of a directory file.
26H	E\$FACCESS	* * * *	File access not granted.
27H	E\$FTYPE	* * *	Incompatible file type.
N	Nucleus Reference Manual		L Loader Reference Manual
B	Basic I/O System Ref Manual		H Human Interface Reference Manual
E	Extended I/O Sys Ref Manual		

CONDITION CODES SUMMARY

Table A-1. iRMX™ 86 Condition Codes (continued)

Hex. Value	Mnemonic	Manuals N B E L H	Meaning
Environmental Conditions (continued)			
28H	E\$SHARE	* * * *	Improper file sharing requested.
29H	E\$SPACE	* *	No space left.
2AH	E\$IDDR	* *	Invalid device driver request.
2BH	E\$IO	* * * *	An I/O error occurred.
2CH	E\$FLUSHING	* * * *	Connection specified in call was deleted before the operation was completed.
2DH	E\$ILLVOL	* * *	Invalid volume name.
2EH	E\$DEV\$OFF- LINE	*	The device being accessed is now offline.
2FH	E\$IFDR	* *	Invalid file driver request.
40H	E\$LOG\$NAME\$- SYNTAX	* *	The specified path starts with a colon (:) but does not contain a second, matching colon.
41H	E\$CANNOT\$- CLOSE	*	The Extended I/O System was not able to transfer remaining data in buffers to output device.
42H	E\$IOMEM	* *	The Basic I/O System has insufficient memory to process a request.
44H	E\$MEDIA	* *	The device containing a specified file is not online.
45H	E\$LOG\$NAME\$- NEXIST	* *	The Extended I/O System was unable to find a specified logical name in the object directories that it checks.
46H	E\$NOT\$OWNER	*	The user who attempted to detach the device is not the owner of the device.
N	Nucleus Reference Manual		L Loader Reference Manual
B	Basic I/O System Ref Manual		H Human Interface Reference Manual
E	Extended I/O Sys Ref Manual		

CONDITION CODES SUMMARY

Table A-1. iRMX™ 86 Condition Codes (continued)

Hex. Value	Mnemonic	Manuals N B E L H	Meaning
Environmental Conditions (continued)			
47H	E\$IO\$JOB	*	The Extended I/O System cannot create an I/O job because the size specified for the object directory is too small.
50H	E\$IO\$UNCLASS	*	An unknown type of I/O error occurred.
51H	E\$IO\$SOFT	* *	A soft I/O error occurred. A retry might be successful.
52H	E\$IO\$HARD	* *	A hard I/O error occurred. A retry is probably useless.
53H	E\$IO\$OPRINT	* *	The device was off-line. Operator intervention is required.
54H	E\$IO\$WRPROT	* *	The volume is write-protected.
60H	E\$ABS\$ADD- RESS	*	An absolute object program was loaded into system protected memory area.
61H	E\$BAD\$GROUP	* *	Invalid group component in the a group definition record.
62H	E\$BAD\$- HEADER	* *	Invalid header record in the object file.
63H	E\$BAD\$SEG- DEF	* *	Invalid segment definition record.
64H	E\$CHECKSUM	* *	A checksum error occurred while reading an object record.
65H	E\$EOF	* *	Unexpected end of file encountered while reading object records.
66H	E\$FIXUP	* *	Invalid fixup record in the object file.
N	Nucleus Reference Manual	L	Loader Reference Manual
B	Basic I/O System Ref Manual	H	Human Interface Reference Manual
E	Extended I/O Sys Ref Manual		

CONDITION CODES SUMMARY

Table A-1. iRMX™ 86 Condition Codes (continued)

Hex. Value	Mnemonic	Manuals N B E L H	Meaning
Environmental Conditions (continued)			
67H	E\$NO\$LOADER \$MEM	* *	Insufficient memory to satisfy loader dynamic memory requirements.
68H	E\$NO\$MEM	* *	Insufficient memory to create PIC/LTL segments.
69H	E\$REC\$FOR- MAT	* *	Invalid record format encountered.
6AH	E\$REC\$- LENGTH	* *	Record length of an object record exceeds configured loader-buffer size.
6BH	E\$REC\$TYPE	* *	Invalid record type encountered in the object file.
6CH	E\$NO\$START	* *	Start address not found.
6DH	E\$JOB\$SIZE	* *	Maximum job-size specified is less than the memory requirement specified in the object file.
6EH	E\$OVERLAY	*	Overlay name does not match with any of the overlay module names.
6FH	E\$LOADER \$SUPPORT	* *	The object file being loaded requires features not supported by the configured loader.
70H	E\$SEG\$ BOUNDS	*	One of the data records in a module loaded by the Application Loader referred to an address outside the segment created for it.
80H	E\$LITERAL	*	The parse buffer contains a literal with no closing quote.
N Nucleus Reference Manual		L Loader Reference Manual	
B Basic I/O System Ref Manual		H Human Interface Reference Manual	
E Extended I/O Sys Ref Manual			

CONDITION CODES SUMMARY

Table A-1. iRMX™ 86 Condition Codes (continued)

Hex. Value	Mnemonic	Manuals N B E L H	Meaning
Environmental Conditions (continued)			
81H	E\$STRING\$- BUFFER	*	The string to be returned as the parameter name exceeds the size of the buffer the user provided in the call.
82H	E\$SEPARA- TOR	*	The parse buffer contains a command separator.
83H	E\$CONTINUED	*	The parse buffer contains a continuation character.
84H	E\$INVALID\$- NUMERIC	*	A numeric value contains invalid characters.
85H	E\$LIST	*	The last value of the value list is missing.
86H	E\$WILDCARD	*	A wild-card character appears in an invalid context, such as an intermediate component of a pathname.
87H	E\$PREPOSI- TION	*	The same preposition as on the the command line was indicated, but can not be used.
88H	E\$PATH	*	The command line specifies an invalid pathname.
89H	E\$CONTROL\$C	*	The user typed CONTROL-C while the command was being loaded.
8AH	E\$CONTROL	*	The command line contains an invalid control.
8BH	E\$UNMATCHED \$LISTS	*	There were no more input pathnames although the output pathname list was not empty.
N	Nucleus Reference Manual		L Loader Reference Manual
B	Basic I/O System Ref Manual		H Human Interface Reference Manual
E	Extended I/O Sys Ref Manual		

CONDITION CODES SUMMARY

Table A-1. iRMX™ 86 Condition Codes (continued)

Hex. Value	Mnemonic	Manuals N B E L H	Meaning
Programmer Errors			
8000H	E\$ZERO\$- DIVIDE	*	A task attempted to divide by zero.
8001H	E\$OVERFLOW	*	An overflow interrupt occurred.
8002H	E\$TYPE	* * * * *	A token parameter referred to an existing object that is not of the required type.
8003H	E\$BOUNDS	*	A task attempted to access beyond the end of a segment.
8004H	E\$PARAM	* * * * *	A parameter which is neither a token nor an offset has an invalid value.
8005H	E\$BAD\$CALL	* *	The I/O System code has been damaged, probably due to a bug in an application task. Recovery is not possible.
8006H	E\$ARRAY\$- BOUNDS	*	Hardware or software has detected an array overflow.
8007H	E\$NDP\$- STATUS	*	An 8087 Numeric Processor Extension error has been detected; Operating System extensions can return the status of the 8087 to the exception handler.
8008H	E\$CHECK\$EX- CEPTION	*	A software interrupt 17 has occurred.
8021H	E\$NOUSER	* * *	No default user.
8022H	E\$NOPREFIX	* * *	No default prefix.
8040H	E\$NOT\$LOG\$- NAME	* *	Specified object is not a device connection or file connection.
N Nucleus Reference Manual B Basic I/O System Ref Manual E Extended I/O Sys Ref Manual		L Loader Reference Manual H Human Interface Reference Manual	

CONDITION CODES SUMMARY

Table A-1. iRMX™ 86 Condition Codes (continued)

Hex. Value	Mnemonic	Manuals N B E L H	Meaning
Programmer Errors (continued)			
8041H	E\$NOT\$- DEVICE	*	A token parameter referred to an existing object that is not, but should be, a device connection.
8042H	E\$NOT\$CON- NECTION	*	A token parameter referred to an existing object that is not, but should be, a file connection.
8060H	E\$JOB\$PARAM	* *	The maximum job-size specified is less than the minimum job-size.
8080H	E\$PARSE\$- TABLES	*	There is an error in the internal parse tables.
8081H	E\$JOB\$- TABLES	*	An internal Human Interface table was overwritten, causing it to contain an invalid value.
8083H	E\$DEFAULT\$SO	*	The default output name STRING is invalid.
8084H	E\$STRING	*	The pathname to be returned exceeds 255 characters in length.
N	Nucleus Reference Manual	L	Loader Reference Manual
B	Basic I/O System Ref Manual	H	Human Interface Reference Manual
E	Extended I/O Sys Ref Manual		

INDEX

Underscored entries are primary references.

:\$: logical name 2-13, 2-19, 3-13

access rights 3-8, 3-26, 3-28, 3-35, 3-46, 3-69, 3-75

accessing the Human Interface 2-4

AD command (files utility) 6-3

AFTER preposition 2-20

ampersand 2-18

AS preposition 2-20

ATTACHDEVICE command 2-10, 3-7

ATTACHFILE command 2-11, 3-13

attaching

- devices 3-7
- files 3-13

automatic search 2-19

BACKUP command 3-17

backup volumes 3-78

:BB: logical name 2-13

beginning a console session 2-4, 4-1

blocks 3-45, 3-49

bootstrap loader 2-3

BR command (files utility) 6-4

byte bucket 2-13

carriage return 1-2

CD command (files utility) 6-4

CHANGEID sub-command 3-87

changing diskettes 6-3

:CI: logical name 2-14

:CO: logical name 2-14

combine-type attributes 5-1

command

- dictionary 3-5
- file 3-83
- line interpreter (CLI) 2-2, 2-4
- name 2-17, 2-19
- syntax 2-17

commands 3-1

comment 2-18

COMMON attribute 5-3

concatenating files 4-6

condition codes A-1

configuration 2-1

connections 2-11, 2-15, 3-14, 3-57

INDEX (continued)

continuation mark 2-18
control characters 1-1
COPY command 3-24, 4-2
CREATEDIR command 3-28, 4-9
creating data files 4-2
creating directories 6-4
CTRL/c 1-5
CTRL/o 1-4
CTRL/p 1-2
CTRL/q 1-4
CTRL/r 1-3
CTRL/s 1-4
CTRL/t 1-4
CTRL/t 1-4
CTRL/u 1-3
CTRL/w 1-4
CTRL/x 1-3

data files 4-2
DATE command 3-29
DC command (files utility) 6-6
DE command (files utility) 6-4
DEBUG command 3-31
default prefix 2-13, 2-14, 4-15
DELETE command 3-33, 4-8, 4-14
deleting files 3-33, 4-8, 6-5
DETACHDEVICE command 3-35
DETACHFILE command 3-38
detaching
 devices 3-35
 files 3-38
 logical names 6-5
device
 logical names 2-10
 name 3-9, 3-63, 3-67
device-unit information block 3-8
DI command (files utility) 6-5
dictionary 3-5
DIR command 2-10, 3-40, 4-13
directories 2-6, 3-28, 3-40, 4-9
discarding mode 1-3, 1-4
diskette switching 4-21, 6-3
DISKVERIFY command 3-49
DOWNCOPY command 3-53
DT command (files utility) 6-5
DUIB 3-8

error messages (Human Interface) 3-1
escape sequences 1-5
examples 4-1
exception codes A-1
EXIT sub-command 3-88
extension data 3-57, 3-59, 4-21, 6-8

INDEX (continued)

file
 data 2-6
 directory 2-6
 length 3-45
 named 2-5, 3-7
 physical 2-6, 3-7
 root directory 2-7
 stream 2-6
 structure 2-5
files 3-57
files utility 6-1
 commands 6-3
 error messages 6-9
 hardware 6-1
 invocation 6-2
fnodes 3-57, 3-59, 6-7
FO command (files utility) 6-7
FORMAT command 2-7, 2-23, 3-56, 4-20
formatting volumes 6-7

global object directory 2-12, 2-13, 3-13, 3-38
granularity 3-45, 3-49, 3-57, 3-60, 4-20, 6-8
groups 3-31

hard-copy mode 1-2
HE command (files utility) 6-9
header records 5-8
hierarchy 2-6
:HOME: logical name 2-14, 3-13
Human Interface commands 3-1

in-place patch 5-6
initial program 2-2, 2-4
INITSTATUS command 3-62
inpath-list 2-17, 2-21
interactive job 2-4, 3-65, 3-67
interleave factor 3-57, 3-60, 4-21, 6-8
internal files 3-59
invisible files 3-41
iSBC 957B package 2-3, 3-31, 3-53, 3-92, 6-1
ISIS-II files 3-53, 3-92, 6-6

job ID 3-64, 3-65
JOBDELETE command 3-65
jump instruction patch 5-5

:LANG: logical name 2-13
library module patching 5-7
line editing 1-1
line feed 1-2
line terminator 1-2, 2-17
link map 3-31
listing directories 4-13, 6-5
listing translator header records 5-8
loading the operating system 2-2

INDEX (continued)

local object directory 2-12, 2-14
LOCK command 3-67
logical names 2-9
 devices 2-10
 files 2-11
logon file 2-5

monitor 2-3, 3-31
multi-access 2-2

named files 2-5, 2-6, 3-7
normal mode 1-3

object directories 2-11
 global 2-12, 2-13, 3-13, 3-38
 local 2-12, 2-14
 root 2-11, 2-13, 3-7
outpath-list 2-17, 2-21
output mode 1-3
OVER preposition 2-20
owner 2-4, 3-35, 3-45, 3-71

parameters 2-17, 2-23
password 3-87
Patching Utility 5-1
 error messages 5-4
 invocation 5-2
 patching procedures 5-5
 versions 5-2
pathnames 2-8
 separators 2-8
PERMIT command 3-69
physical files 2-6, 3-7
physical names 3-9
prefix 2-10, 2-13, 2-14, 2-19
preposition 2-17, 2-20
:PROG: logical name 2-14, 2-19
:PROG:R?LOGON file 2-5
prompt 2-5, 3-87
PTCH86.86 5-2
PTCH86.R86 5-2
PUBLIC attribute 5-3

quoting characters 1-3, 2-18

R?BADBLOCKMAP file 3-59, 6-8
R?FNODEMAP file 3-59, 6-8
R?LOGON file 2-5
R?SPACEMAP file 3-59, 6-8
removing volumes 2-15
RENAME command 3-74, 4-14, 4-16
replacement modules 5-1
replacing files 4-5
requirements 2-1
RESTORE command 3-77

INDEX (continued)

RMX86 file 2-3
root directory 2-7, 2-10, 3-59
root object directory 2-11, 2-13, 3-7
rubout 1-2

scrolling mode 1-3, 1-4
:SD: logical name 2-13
search order 2-19
segments 3-31, 5-3
separators 2-8
single-access 2-2
state 3-64
stopped mode 1-3, 1-4
storing logical names 2-11
stream files 2-6
:STREAM: logical name 2-13
structure of files 2-5
sub-commands 3-87
SUBMIT command 2-5, 3-83
SUPER command 3-87
switching diskettes 4-21
syntax 2-17, 3-3
system device 2-13
:SYSTEM: logical name 2-13, 2-19, 3-1
system manager 2-24, 3-8, 3-28, 3-35, 3-87
SYSTEM/RMX86 file 2-3

terminal device name 3-63, 3-67
Terminal Handler 1-1
Terminal Support Code 1-1
TIME command 3-90
TO preposition 2-20
translator header records 5-8
trees 2-6
type-ahead 1-1

UC command (files utility) 6-9
UPCOPY command 3-92
user ID 2-4, 2-24, 3-46, 3-64, 3-87, 4-1
user state 3-64
:UTILS: logical name 2-13

VERSION command 3-95
volume 2-7, 2-15, 3-77
 boundaries 4-18
 name 3-57, 3-58, 6-7
wild cards 2-15, 4-4

:WORK: logical name 2-13
WORLD 2-4, 3-35, 3-8



REQUEST FOR READER'S COMMENTS

Intel's Technical Publications Departments attempt to provide publications that meet the needs of all Intel product users. This form lets you participate directly in the publication process. Your comments will help us correct and improve our publications. Please take a few minutes to respond.

Please restrict your comments to the usability, accuracy, readability, organization, and completeness of this publication. If you have any comments on the product that this publication describes, please contact your Intel representative. If you wish to order publications, contact the Intel Literature Department (see page ii of this manual).

1. Please describe any errors you found in this publication (include page number).

2. Does the publication cover the information you expected or required? Please make suggestions for improvement.

3. Is this the right type of publication for your needs? Is it at the right level? What other types of publications are needed?

4. Did you have any difficulty understanding descriptions or wording? Where?

5. Please rate this publication on a scale of 1 to 5 (5 being the best rating). _____

NAME _____ DATE _____

TITLE _____

COMPANY NAME/DEPARTMENT _____

ADDRESS _____

CITY _____ STATE _____ ZIP CODE _____
(COUNTRY)

Please check here if you require a written reply.

WE'D LIKE YOUR COMMENTS . . .

This document is one of a series describing Intel products. Your comments on the back of this form will help us produce better manuals. Each reply will be carefully reviewed by the responsible person. All comments and suggestions become the property of Intel Corporation.



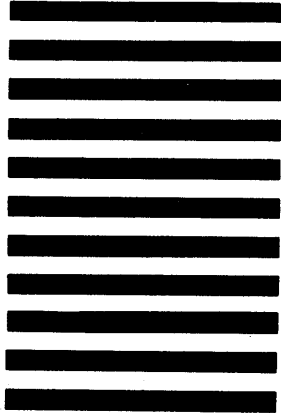
**NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES**

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 79 BEAVERTON, OR

POSTAGE WILL BE PAID BY ADDRESSEE

**Intel Corporation
5200 N.E. Elam Young Pkwy.
Hillsboro, Oregon 97123**

OMO Technical Publications





INTEL CORPORATION, 3065 Bowers Avenue, Santa Clara, California 95051 (408) 987-8080

Printed in U.S.A.