

iRMX 86™ HUMAN INTERFACE REFERENCE MANUAL

Order Number: 9803202-02

REV.	REVISION HISTORY	PRINT DATE
-001	Original Issue	5/81
-002	Adds information about the Disk Verification Utility, the BACKUP and RESTORE commands, and exceptional condition codes; changes information about the SUBMIT and FORMAT commands; corrects technical and typographical errors; and documents Release 4 of the iRMX 86 Operating System.	11/81

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

Literature Department
Intel Corporation
3065 Eowers 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 the 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:

BXP
CREDIT[™]
i
ICE
iCS
im
INSITE
Intel
Intel

Intelevison
Intellec
iRMX
iSBC
iSBX
Library Manager
MCS
Megachassis
Micromainframe

Micromap
Multibus
Multimodule
Plug-A-Bubble
PROMPT
Promware
RMX/80
System 2000
UPI
μScope

and the combination of ICE, iCS, iRMX, iSBC, iSBX, MCS, or RMX and a numerical suffix.

PREFACE

This manual is the primary reference for using the iRMX 86 Human Interface software on your iRMX 86 Operating System. The manual is addressed to two levels of users:

- Operators who will be using Intel-supplied Human Interface commands and (optionally) user-created commands at the console keyboard.
- Development programmers who will be writing custom application programs that can be loaded and executed via interactive keyboard commands.

The manual is implicitly divided into two general sections that approximate these two Human Interface user-levels:

1. Chapters 1 through 4 are addressed to all levels of readers and describe the syntax, format, and uses of the supplied Human Interface commands:
 - Chapter 1 introduces you to the purpose and scope of the Human Interface.
 - Chapter 2 describes the general syntax rules and other considerations for using interactive commands to load and execute programs during your console sessions.
 - Chapter 3 describes the function and syntax of Human Interface commands, plus the interactive system prompts and user responses that take place during command execution.
 - Chapter 4 provides a series of Human Interface command examples that introduce the console user to basic iRMX 86 file management.
2. Chapters 5 through 7 are addressed to the programmer who will be writing new application commands that can be executed from the console keyboard under the control of the Human Interface:
 - Chapter 5 describes some of the programming considerations for using Human Interface system calls in a program's source code.
 - Chapter 6 describes the Human Interface system calls in alphabetical sequence. A system call dictionary is provided at the beginning of the chapter for fast reference.

PREFACE (continued)

- Chapter 7 provides an example of how Human Interface calls are used to create a new command.
- Appendixes A, B, and C describe the Human Interface type definitions, exception codes, and string table format, respectively.

RELATED PUBLICATIONS

The following manuals provide additional background and reference information:

<u>Manual</u>	<u>Number</u>
Introduction to the iRMX 86™ Operating System	9803124
iRMX 86™ Nucleus Reference Manual	9803122
iRMX 86™ Terminal Handler Reference Manual	143324
iRMX 86™ Basic I/O System Reference Manual	9803123
iRMX 86™ Extended I/O System Reference Manual	143308
iRMX 86™ Loader Reference Manual	143318
iRMX 86™ System Programmer's Reference Manual	142721
iRMX 86™ Configuration Guide	9803126
iRMX 86™ Programming Techniques	142982
iRMX 86™ Disk Verification Utility Reference Manual	144133
8086 Family Utilities User's Guide for 8080/8085-Based Development Systems	9800639
iAPX 86,88 Family Utilities Users' Guide for 8086-Based Development Systems	121616
Users' Guide for the iSBC 957B™ iAPX 86,88 Interface and Execution Package	143979
Guide to Writing Device Drivers for the iRMX 86™ and iRMX 88™ I/O Systems	142926
iMMX 800 Software Reference Manual and User's Guide	143808

CONTENTS

	PAGE
CHAPTER 1	
INTRODUCTION	
Human Interface Software Requirements.....	1-1
Human Interface Services.....	1-2
Human Interface Command Set Features.....	1-3
Human Interface System Call Features.....	1-3
File And Directory Structures.....	1-4
Files.....	1-4
Pathname.....	1-4
Directories.....	1-5
Supplied Directories.....	1-5
User Directories.....	1-6
Volumes.....	1-7
Logical Names.....	1-7
Summary.....	1-8
CHAPTER 2	
HUMAN INTERFACE COMMAND ENTRY AND SYNTAX	
Console Session.....	2-1
Command Line.....	2-2
Additional Command Syntax.....	2-3
Command Line Syntax Example.....	2-3
Keyboard Character Entries.....	2-4
Input And Output Parameters.....	2-5
Pathlists.....	2-5
Preposition Parameters.....	2-6
Control Keys.....	2-7
File Handling Considerations.....	2-8
Directory Listings.....	2-9
File Handling Safeguards.....	2-9
CHAPTER 3	
HUMAN INTERFACE COMMANDS	
Command Syntax Schematics.....	3-1
ATTACHDEVICE.....	3-5
BACKUP.....	3-8
COPY.....	3-15
CREATEDIR.....	3-18
DATE.....	3-20
DEBUG.....	3-21
DELETE.....	3-22
DETACHDEVICE.....	3-24
DIR.....	3-25
DISKVERIFY.....	3-32
DOWNCOPY.....	3-37
FORMAT.....	3-40
RENAME.....	3-45
RESTORE.....	3-48

CONTENTS (continued)

	PAGE
CHAPTER 3 (continued)	
SUBMIT.....	3-55
TIME.....	3-58
UPCOPY.....	3-59
CHAPTER 4	
FILE HANDLING 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 Duplicate Files.....	4-3
How To Copy To New Files.....	4-4
How To Display Files.....	4-4
How To Replace Existing Files	4-5
How To Concatenate Files.....	4-6
How To Delete Files.....	4-8
How To Use Directories.....	4-8
How To Create A New Directory.....	4-9
How To Reference A Directory.....	4-10
How To Add New Entries To A Directory.....	4-10
How To Create A Directory Within A Directory.....	4-11
How To List Directories.....	4-12
How To Move Files Between Directories.....	4-13
How To Delete A Directory.....	4-13
How To Rename Files And Directories.....	4-14
How To Rename Files.....	4-14
How To Rename Directories.....	4-16
How To Move Files Across Volume Boundaries.....	4-16
How To Format A New Volume.....	4-18
Diskette Switching Procedures.....	4-19
How To Use A Submit File.....	4-20
CHAPTER 5	
CREATING NEW COMMANDS	
General Command Line Structure.....	5-1
Input And Output Parameter Calls.....	5-2
Message Processing Calls.....	5-4
Command Processing Calls.....	5-5
Program Control Call.....	5-5
Parameter Parsing Calls.....	5-5
Parameter Syntax Considerations.....	5-6
Command Invocation.....	5-7
Program Control.....	5-7
Exception Handler.....	5-8
Logical Names.....	5-8
Load Module Formats.....	5-8
Command Creation Procedures.....	5-9

CONTENTS (continued)

	PAGE
CHAPTER 6	
HUMAN INTERFACE SYSTEM CALLS	
C\$CREATE\$COMMAND\$CONNECTION.....	6-4
C\$DELETE\$COMMAND\$CONNECTION.....	6-10
C\$FORMAT\$EXCEPTION.....	6-11
C\$GET\$CHAR.....	6-13
C\$GET\$INPUT\$CONNECTION.....	6-14
C\$GET\$INPUT\$PATHNAME.....	6-20
C\$GET\$OUTPUT\$CONNECTION.....	6-22
C\$GET\$OUTPUT\$PATHNAME.....	6-31
C\$GET\$PARAMETER.....	6-34
C\$SEND\$COMMAND.....	6-38
C\$SEND\$CO\$RESPONSE.....	6-48
C\$SEND\$EO\$RESPONSE.....	6-51
C\$SET\$CONTROL\$C.....	6-54
C\$SET\$PARSE\$BUFFER.....	6-56
CHAPTER 7	
COMMAND CREATION EXAMPLES	
Example 1. Standard System Calls.....	7-2
Example 2. Parsing Call.....	7-3
Example 3. Advanced System Calls.....	7-4
APPENDIX A	
TYPE DEFINITIONS.....	A-1
APPENDIX B	
HUMAN INTERFACE EXCEPTION CODES.....	B-1
APPENDIX C	
STRING TABLE FORMAT.....	C-1

CONTENTS (continued)

	PAGE
FIGURES	
3-1. EXTENDED Directory Listing Example.....	3-25
3-2. FAST Directory Listing Example.....	3-25
3-3. LONG Directory Listing Example.....	3-26
3-4. SHORT Directory Listing Example.....	3-26
C-1. String Table Format.....	C-1

TABLES	
3-1. Human Interface Command Dictionary.....	3-3
3-2. Directory Listing Headings.....	3-27
5-1. Standard Logical Names.....	5-8
6-1. System Call Dictionary.....	6-2
A-1. Type Definitions.....	A-1
B-1. Human Interface Exception Codes.....	B-1
B-2. Exception Code Ranges.....	B-2
B-3. iRMX 86™ Condition Codes.....	B-3

CHAPTER 1. INTRODUCTION

The iRMX 86 Human Interface allows console operators to load and execute program files from a console keyboard. The operator types a command name that is the name of the application program to be executed. That program is loaded into main memory from secondary storage and the program is executed in conformance with any required or optional parameters that the operator specified in the command line.

The console operator and the Human Interface begin an interactive dialog once a command is typed. A Human Interface module called the Command Line Interpreter (CLI) scans the command line for validity and then executes the associated program. If any syntax or other errors are detected in the command entry, an explanatory error message is displayed on the console screen and the operator is prompted for a new command entry.

During execution, further dialog between the console operator and the program itself may take place if the program encounters specifications in the command line that require further clarification. The program then prompts the operator for a course of action it is to follow. Generally, such queries are issued because a command line parameter is subject to ambiguous interpretation, or to protect existing program or data file structures from accidental destruction or modification. The displayed prompts and queries are coded into the program via Human Interface system calls when the program is written.

When program execution is completed, a status message is displayed that defines what action(s) the program performed. The Human Interface then prompts for a new command entry. Thus, program execution is truly interactive, from the time the operator enters a command until execution is completed. The operator controls and sometimes modifies execution through command parameter entries and keyboard responses to prompts issued by the program. Human Interface functions, in turn, respond to the operator keyboard entries by displaying status messages, error messages or prompts to ensure successful program operation.

HUMAN INTERFACE SOFTWARE REQUIREMENTS

The Human Interface functions can be used on any iRMX 86 Operating System that is configured with all of the following software layers:

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

INTRODUCTION

- Application Loader
- Human Interface

The Human Interface is a software layer in an iRMX 86 Operating System that is configured with the above-named modules. During command execution, the Human Interface invokes the services of these software modules in a way that is transparent to the console operator. Therefore, an operator needs little or no knowledge of operating system structures to load and execute programs from the console keyboard.

HUMAN INTERFACE SERVICES

The Human Interface provides and supports two distinct levels of services for iRMX 86 users:

1. A set of non-resident Human Interface commands that consist of file management and general utility routines. These commands are available for immediate use on any iRMX 86 Operating System that is configured with the Human Interface. The Human Interface commands are described in detail in Chapters 3 and 4 of this manual.
2. A set of system calls that provides programmers with convenient routines for creating new applications commands. This feature is of particular value to OEM's who are writing custom iRMX 86 applications for their own customers. The programmer invokes specific command handling functions associated with the new program by including the relevant Human Interface calls in the program's source code.

Once a new command is assembled and linked, the programmer stores it either under the :PROG: directory in secondary storage, or under the :SYSTEM: directory for debugged commands (see "Supplied Directories" later in this chapter for a description of the :PROG: and :SYSTEM: directories). The Human Interface calls are described in Chapters 5 and 6.

All iRMX 86 Nucleus, I/O System, Extended I/O System, and Application Loader calls are also available when writing an application program associated with a user-designed command. See the appropriate iRMX 86 reference manuals.

Both the Intel-supplied and user-created commands should use the same general syntax. This consistency in syntax makes it easier for the console operator to use commands at the keyboard and aids the system programmer who is designing and creating new commands.

INTRODUCTION

HUMAN INTERFACE COMMAND SET FEATURES

Some of the main features provided by the Intel-supplied file handling and utility routines are described in the following list. The console operator can invoke any of the services by entering a single command:

- Create a new file.
- Create new directories.
- List the contents of a file.
- List a directory, plus the attributes of all files in the directory. Five listing formats are available, depending upon the amount of information desired.
- Rename files or directories across directory boundaries.
- Delete files and/or directories.
- Append one or more files to the end of an existing file, or concatenate files.
- Copy files from an ISIS-II volume mounted on a secondary storage device to an iRMX volume mounted on a secondary storage device, or vice versa.
- Read command input from a secondary storage device.
- Format a new volume.
- Set or reset the system date and time from the console keyboard.

HUMAN INTERFACE SYSTEM CALL FEATURES

The Human Interface system calls let programmers create new commands that are compatible with the standard syntax. The calls consist of command parsing, processing, and I/O connection routines that link an application program with iRMX 86 Operating System resources in a way that is completely transparent to the operator during command execution.

Optional parameter elements in the calls give the programmer the ability to create new commands that are tailored to the exact requirements of the end user. An important feature of the calls is that they ensure a consistent syntax and processing for new commands. Thus, user-created commands will still be usable without modification or rewriting in future releases of the Human Interface.

Some of the main services provided by the Human Interface system calls are as follows:

- Define the location of the user's system console input and output devices (:CI: and :CO:, which are described later in the manual).

INTRODUCTION

- Set up I/O connections to input and output files specified in a command line.
- Set up a command connection to send multiple-line commands.
- Allow a user to obtain control of the CTRL/C key.
- Concatenate multiple-line commands and then execute the command.
- Format a default message for a given exception code.
- Send and receive messages to and from the user's console.

FILE AND DIRECTORY STRUCTURES

The following subsections in this chapter define the file organization and structures used both by the console operator and by executing programs.

FILES

The file is the basic unit in the file organization. A file has a pathname by which it is accessed, and it resides on a volume of some secondary storage device rather than in main memory (see the pathname definition under the next heading). A named file may consist of data accessed by one or more programs, data that you access at the console, a directory, or an executable program file that you invoke by a Human Interface command. For example, all the supplied Human Interface commands are non-resident program files.

PATHNAME

A pathname is a "road map" through the hierarchical tree structure for a named file or directory that is to be accessed during command execution. This pathname tells the executing command which directories (and in some cases which device; see "Volumes" in this chapter) to search to find the desired file or directory.

When you list a pathname in a command entry, enter each name element within the path in a descending hierarchical order, and separate each of these pathname elements with a slash mark (/). For example:

```
:F1:PRIME/LEVELA/mytest
```

is the pathname for a file named "mytest" which is located on a volume mounted on a device whose prefix is :F1:. PRIME/LEVELA means the command is to search a parent directory named PRIME for a another directory named LEVELA, which in turn, is searched for the mytest file to be accessed. Note that the file or directory to be read from or written to is always the last element in the specified path.

INTRODUCTION

All of the following are valid pathnames:

a	(located on system's default directory)
able	(located on system's default directory)
:fl:/able/samp	(Pathname for samp located in able directory on the device with prefix :Fl:)

Each time you access a file or directory for a read or write, you must enter the complete pathname. Once you create a new file or directory, its pathname remains fixed unless you choose to rename or move that file or directory to some new path. Note that pathname entries can be typed in either uppercase or lowercase letters.

DIRECTORIES

A directory is a specialized file used to store a list of file names and (optionally) other directory names in logical groupings. What distinguishes a directory from other files is that a directory maintains a formatted list of all subordinate directories and files that are created and assigned under its name. This formatted list of file and directory pathnames can be either displayed on the operator's console screen or written to a specified output file via the DIR command.

A directory name is always a preceding component in the pathnames of files or other directories listed under that directory. You can display any directory's list of files and other directories by specifying the directory's pathname in a DIR command (see Chapter 3).

There are two different types of directories: those supplied with the Human Interface, and those you create yourself by using the CREATEDIR command.

SUPPLIED DIRECTORIES

Intel supplies four directories with the Human Interface software. These directories each have a logical name enclosed by colon characters. When you wish to specifically list one of the directories, this logical name must be specified in the DIR command. The logical names and purpose of the supplied directories are as follows:

:PROG:	The user's program directory in which all user-created commands can be placed. When you type any command on the console keyboard, this is the first directory that the Human Interface automatically searches for that command.
:SYSTEM:	The directory in which the Intel-supplied Human Interface commands are placed. When a command is typed on the console keyboard, the Human Interface searches this directory for that command <u>after</u> it has searched the :PROG: directory.

INTRODUCTION

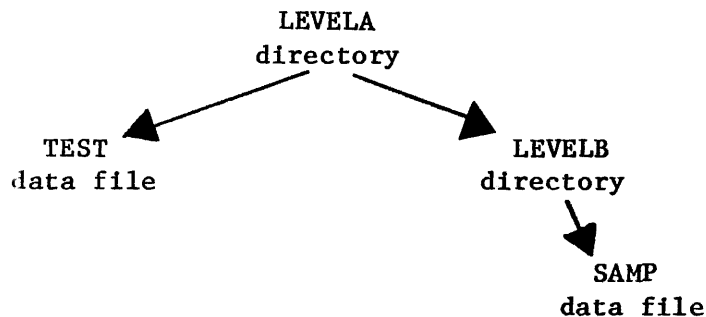
- :\$:** The prefix for the default directory on the system. If you do not specifically list some other directory in the pathname of files and directories you create and subsequently access during console sessions, they will be assigned to and listed from the default directory. The **:\$:** prefix need not be specified in a command.
- :WORK:** The user's work directory, which is a temporary or "scratch" work area. This directory can be used either programmatically by an executing command or by a user during a console session. Files in this directory are highly volatile once detached..

All the supplied directory assignments described previously are configurable options during system configuration time, including the order of search for command names in the **:PROG:** and **:SYSTEM:** directories.

USER DIRECTORIES

User directories are those you create during console sessions to expedite the management of your files. Once you create a new directory by using the **CREATEDIR** command (see Chapter 3), that directory's pathname becomes the preceding element in the pathname hierarchy for any file you subsequently assign to the directory's maintained list of files.

For example, if a directory named **LEVELA** contains a directory named **LEVELB** and a data file named "test", the pathnames for **LEVELB** and **test** would be **LEVELA/LEVELB** and **LEVELA/test** respectively. Similarly, if the **LEVELB** directory contains a file named **SAMP**, the heirarchical structure for this tree would be as follows:



and all accesses to **SAMP** would require the following pathname:

`levela/levelb/samp`

Note that in the above example, the directory and file pathnames were assigned to the system's default user directory. All directories you create will automatically be assigned and listed in your system's default directory unless you specify the prefix for some other directory as the preceding element in the pathname of the new directory.

INTRODUCTION

VOLUMES

A volume is a secondary storage device, such as a diskette, hard disk platter, or a bubble memory that you have formatted to accept files and directories. Before you can enter any file or directory on a new volume, that volume must be formatted by using the Human Interface FORMAT command (see the FORMAT command description in Chapter 2).

To access a volume mounted on a device other than the default device for your configuration, you specify the device's prefix as the first element in the pathname of the file or directory that is being created or accessed on the volume in that device. The prefix is actually the logical name for the device on which the volume is mounted; for example, :F1:. See the FORMAT command in Chapter 3 and the volume access examples in Chapter 4 for more detailed information.

LOGICAL NAMES

Logical names let you refer to connections symbolically. You differentiate a logical name from a component of a pathname by bracketing the logical name with colons. You can use a logical name either as a prefix or in place of a pathname.

The standard logical names and their meaning are described in Table 2-1. However, note that these logical name assignments are all configurable during system configuration time. See the iRMX 86 CONFIGURATION GUIDE for more information.

You can also assign logical names to new physical devices that are added to the system without having to go through a system reconfiguration. See the ATTACHDEVICE command in Chapter 3 for more information.

Table 2-1. Standard Logical Names

Name	Meaning
:CI:	Prefix for user console input (default input file)
:CO:	Prefix for user console output (default output file)
:PROG:	Prefix to User Program Directory
:SYSTEM:	Prefix to System Command Directory)
:\$:	Default prefix
:WORK:	Prefix to User Work Directory
:BB:	Byte bucket

INTRODUCTION

The full meaning of the logical names listed in Table 2-1 is as follows:

- :CI:** and **:CO:** Logical devices that establish the source of commands and data and the command destination output, respectively.
- :PROG:** The directory where the user-designed commands are located. The directory name represented by **:PROG:** is a configuration option.
- :SYSTEM:** The directory where the Human Interface commands are located. The directory name represented by **:SYSTEM:** is a configuration option.
- :\$:** The user's default directory, which is used for files that do not have a logical name. This directory is a configuration option.
- :WORK:** The directory to be used for temporary files and work ("scratch") files. The directory name represented by **:WORK:** is a configuration option.
- :BB:** The logical name of a device that is treated as an infinite sink. Anything written to **:BB:** will disappear, and a read from **:BB:** will receive an EOF (end-of-file).

SUMMARY

During a Human Interface console session, a operator accesses system services interactively rather than programmatically. At this level, the dialog from operator to system is through keyboard commands, parameter options, and responses to queries or prompts. The dialog from system to operator is through displayed prompts, queries, status and error messages, and listed output.

At the programming level, the Human Interface provides routines that allow programmers to create new commands whose syntax and structure is consistent with the features and requirements of the current and future releases of the Human Interface.

CHAPTER 2. HUMAN INTERFACE COMMAND ENTRY AND SYNTAX

This chapter describes the general syntax rules and procedures to be followed when using Human Interface commands at an operator's console. Additional and more specific information is provided in the individual command descriptions in Chapter 3.

When you enter a command at the console keyboard, the associated program file is loaded and executed in conformance with any parameters you included in the command. During execution, the program may prompt or query you for more information as to the precise operation you wish performed. Following execution, the Human Interface displays status messages that tell you the nature and scope of the operations performed and whether or not execution was successful.

Error messages may also be displayed if you attempt an invalid operation (e.g., a command syntax error) or if some error is encountered while the command is being executed. Human Interface error messages are defined in the individual command descriptions in chapter 3.

The general syntax rules described in this chapter apply equally to both the supplied Human Interface commands and any user-created commands that may have been added to your system.

CONSOLE SESSION

A console session begins when you enter a command at the the console keyboard to load and execute a program. When your iRMX 86 Operating System is initialized, the Human Interface displays the following message to let you know that it has initialized and is ready to accept your first keyboard command entry:

```
iRMX 86 HI Vx.x: user = WORLD
```

```
-
```

where:

- | | |
|--------------|--|
| Vx.x | Is the number of the version of the Human Interface that is currently running on your system. |
| user = WORLD | Means you have read and write access to all files and directories on the system. |
| - (hyphen) | Prompts you to enter a command. After command execution is completed or is prematurely terminated for any reason, the Human Interface displays another hyphen to prompt for your next command. |

HUMAN INTERFACE COMMAND ENTRY AND SYNTAX

Enter your command on the same line and immediately following the hyphen prompt. For example:

```
-copy a to b
```

where everything following the hyphen is your command entry. Type a carriage return to end a command line and to position the cursor on a new line. In all examples in this manual, it is assumed that you will terminate every command line or data line by pressing the RETURN, LINE FEED, or ESC key.

COMMAND LINE

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 that executes a file handling routine is as follows:

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

where:

- | | |
|--------------|---|
| command name | Pathname of the non-resident program file to be executed. After the command line is entered, the named program file is loaded into main memory from secondary storage for execution. |
| inpath-list | One or more pathnames for the files to be read as input during command execution. Multiple pathnames in an input file list must be separated by commas. Embedded blanks between pathnames are optional. See the description of pathlists later in this chapter. |
| preposition | A word that tells the executing command how you want the output handled. The four prepositions used in Intel-supplied commands are TO, OVER, AFTER, and AS. Your system may also have other prepositions for user-created commands. See the descriptions of prepositions later in this chapter. |
| outpath-list | One or more pathnames for the files that are to receive the output during command execution. Multiple pathnames specified in an output file list must be separated by commas. Embedded blanks between pathnames are optional. |
| parameters | Specifically requested or defaulted optional parameters that perform additional or extended services during command execution. The QUERY parameter that is available in some Human Interface commands, or the EXTENDED parameter in the DIR command are examples of such parameters. See the individual parameter descriptions for the commands in Chapter 3. |

HUMAN INTERFACE COMMAND ENTRY AND SYNTAX

ADDITIONAL COMMAND SYNTAX

line terminator Not shown in the command line format, but terminates the command entry on the current line and causes the cursor to go to a new line. The RETURN key, NEW LINE key, or ESC key are used as line terminators, depending on your hardware configuration.

Other optional command line entries are as follows:

continuation mark An ampersand character (&) to indicate the command line is continued on the next line. When you press the RETURN key after you press the ampersand key, the Human Interface displays two asterisks (**) on the next line to prompt for the continuation line.

Within available memory limits, you can use as many continuation lines for a given command as you desire. After you press the RETURN key without a preceding ampersand character, the invoked command will receive the entire command string as a single command.

Note that an ampersand within a quoted string will not be interpreted as a continuation mark.

comment character A semicolon (;) character to indicate that all text following it on the current line is to be read as a non-executable comment. Comments can also be entered after a continuation mark (&) is entered on a line but before the RETURN key is pressed. A common use of comments in commands is in a SUBMIT file list of commands (see the SUBMIT command in Chapter 3).

Note that a semicolon within a quoted string will not be interpreted as a comment character.

A command line and each succeeding continuation line for a complete command can have a maximum of 255 characters per line, including any punctuation, embedded blanks, continuation mark, non-executable comments, and carriage return.

The Command Line Interpreter prescans the current line for a continuation mark each time you press the RETURN key. If a continuation mark is present, the cursor is positioned to a new line and the continuation prompt (**) is issued. If a continuation mark is not present in the line when the RETURN key is pressed, the command is loaded and executed.

COMMAND LINE SYNTAX EXAMPLE:

The following example incorporates all of the command elements described in the previous general syntax subsection except a comments entry:

HUMAN INTERFACE COMMAND ENTRY AND SYNTAX

```
-copy test/sampl/data, test/more/prog to &  
**new/file1,new/file2 query
```

where:

-	Human Interface prompt for a new command entry.
copy	Name of the command to be executed.
test/sampl/data	First pathname in the input pathlist.
,	Pathname separator between input pathnames (embedded blank is optional).
test/more/prog	Second (and last) pathname in the input list.
to	Preposition that tells the command to copy the input files TO new files.
&	Continuation character that tells the Human Interface you wish to continue the command on the next line.
RETURN (not shown)	Terminates the command entry on this line and causes the cursor to go to a new line.
**	Human Interface prompt to continue the command entry on the new line.
new/file1	Pathname of the first file in the output list to receive the output from the corresponding file in the input list.
,	Separator between pathnames.
new/file2	Pathname of the next file in the output list to receive the output from the second input file.
query	Optional parameter that causes the command to prompt for permission to copy each input file to each listed output file in the listed sequence.
RETURN (not shown)	Ends the command entry and causes the command to be loaded from secondary storage for execution. The screen cursor is positioned on a new line.

KEYBOARD CHARACTER ENTRIES

You can type 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, and your entries will appear on the screen exactly as you enter them. For example, you could create a new file with the pathname "MY/TEST" and then specify the file as "my/test" in subsequent file accesses.

INPUT AND OUTPUT PARAMETERS

An input parameter consists of a pathname or pathname-list of files on which a command is to operate. An output parameter defines the destination or destinations of the processed output. Input and output parameters may consist of the pathnames for one or more files or directories, or logical names, where a logical name is a shorthand way (prefix) of referring either to some physical device configured into your system or to a file. The prefix is always enclosed by colons; for instance, :F1: or :PROG:.

An input parameter is required for all file handling commands. An output parameter is also generally required but the default TO :CO: (to console screen) can be used in most commands. See the individual command descriptions in Chapter 3 for a description of default parameters.

PATHLISTS

A pathlist is a list of pathnames, separated by commas, that comprises an input or output parameter. A command reads the pathnames in a pathlist in the sequence in which you enter them. Such lists of pathnames are referred to in this manual as "inpath-lists" and "outpath-lists" respectively. The example:

```
...PRIME/urfile,LEVEL3/samp, LEVEL5/TEST/one...
```

is a valid pathlist consisting of three pathnames, two of which are separated by a comma, and the last separated by both a comma and a separating blank.

The Human Interface also recognizes your console keyboard and console screen as an "input file" and an "output file" respectively, and are specified in a command line as follows:

:CI: The logical name prefix for the console input device (console keyboard). Specifying :CI: in a command line tells the command to read input from the console keyboard.

:CO: The logical name prefix for the console output device (console screen). For some commands such as COPY, TO :CO: is the default output file if a preposition and output file are not specified. For example, the command:

```
copy test
```

causes the COPY command to write the contents of file TEST TO the console screen by default.

PREPOSITION PARAMETERS

Preposition parameters in a command line tell the the command how you want it to process the output file or files. The Human Interface file management commands usually provide three options in the choice of a preposition: TO, OVER, and AFTER, with AS as yet another preposition used in the ATTACHDEVICE command. 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 does already exist, the command will display the following query on the console screen:

pathname, already exists, DELETE?

Enter a Y or y if you wish to delete the existing file. Enter any other character if you do not wish the file to be deleted. If a Y is not entered, the executing command will not process the corresponding input file but will go to the next input file in the command line. The listed input files are processed and written to the output files on a one-for-one basis in the listed sequence. For example:

-copy a,b to c,d

where file a is copied to file c, and file b is copied to file d.

OVER Causes the command to write your input files over the output files in the listed sequence, irrespective of whether or not the output files already exist. For example:

-COPY sampl,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

causes the contents of file IN1 to be written to the end of the contents of DEST1, and the contents of IN1 to be added to the end of DEST2.

HUMAN INTERFACE COMMAND ENTRY AND SYNTAX

AS Causes a physical device specified in an ATTACHDEVICE command to be cataloged in the system's logical-name directory AS the logical name specified in the command. All subsequent accesses to the device, either interactively by keyboard command or programmatically, must be performed by specifying the cataloged logical name unless that logical name is configured into the system as the standard default prefix.

CONTROL KEYS

CONTROL keys allow you to enter control commands to the Terminal Handler. You invoke control commands by pressing the CTRL key, and while holding it down, also pressing the required alphabetical key. In this manual, CONTROL key functions are designated as follows:

CTRL/character

where CTRL specifies the CONTROL key, and character is the alphabetic character used to perform the desired action. The CTRL keys and their actions are as follows:

CTRL/z	Signals end-of-file for keyboard entry applications. CTRL/z should be entered on a new line because a RETURN or LINE FEED will not be appended to the end of the last data line.
CTRL/c	Tells the Human Interface to abort the currently executing program.
CTRL/d	Invokes the iRMX 86 Debugger (if configured in your system).
CTRL/o	Suppresses output if output is in normal mode, or restores output to normal mode if output is already suppressed.
CTRL/s	Suspends writing to the output file.
CTRL/q	Resumes writing to the output file.
CTRL/x	Deletes the current displayed line during keyboard input.
CTRL/r	Operates in two modes: if the current line is empty, CTRL/r sends the previous line to the console output device so that you can edit and submit the line as a new command; if the current line is not empty CTRL/r sends a RETURN to the console output device, followed by a reentry of the current line.

HUMAN INTERFACE COMMAND ENTRY AND SYNTAX

Other key operations that can be used are as follows:

RUBOUT	Permits simple line editing on the current line. The handling depends upon how the Terminal Handler option was configured in your system. In one option, each time the RUBOUT key is pressed, the last displayed character is deleted with the cursor moving backward one space across the line. You continue pressing the RUBOUT key until you reach the character to be corrected. In the other option, the deleted character is echoed back to the console output device.
RETURN, ESC, or LINE FEED	Signals the end of a command line or line of data and positions the cursor on a new line. In this manual, use of the RETURN key is assumed in command examples.

FILE HANDLING CONSIDERATIONS

Several of the Human Interface commands require that you specify a preposition parameter in the command line. That is, you must enter a TO, OVER, or AFTER preposition as one of the command line parameters. In such cases, the number of input files you list, as opposed to the number of output files, can directly affect the way in which the command handles the preposition and the way the files are processed.

Usually (but not necessarily), you will be specifying a one-for-one match between the number of input files and output files. There are situations where you can combine preposition functions.

For example, in a command (other than RENAME), if you specify more pathnames in the input list than in the output list, the remaining input files will be automatically appended to the end of the last specified output file, regardless of what preposition you specified.

This flexibility in file handling 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. For instance, assume that in a COPY command, you use the TO preposition and specify the following file names in the input and output parameters:

```
copy a,b,c TO d,e
```

When you execute the command line, file "a" will be copied to file "d", and files "b" and "c" will be appended to the end of file "e" as follows:

```
a TO d
b TO e
c AFTER e
```


HUMAN INTERFACE COMMAND ENTRY AND SYNTAX

Conversely, if you specify fewer file names in the input list than in the output list of a command line, the excess output file names will be ignored, regardless of which preposition you specify. For example, assume that in a command line you specify the following file names in the input and output parameters:

```
copy a,b TO d,e,f,g
```

When the command is executed, file "a" will be matched with file "d", file "b" matched with file "e", and files "f" and "g" will be ignored, as follows:

```
a TO d
b TO e
```

Preposition parameters cannot be combined in a RENAME command; that is, TO or OVER prepositions do not convert to an AFTER preposition if there are more input files than output files.

DIRECTORY LISTINGS

A directory listing contains a list of files and other directories that you assign to it. You can list a directory on the console screen or to a named output file by using the DIR command and specifying the directory's pathname (if you omit a directory name in a DIR command, the default user directory will be listed). The amount of information provided for each listed file or directory, and the format of the directory listing depend on which parameter option you specify in the DIR command. See the DIR command description in Chapter 3.

FILE HANDLING SAFEGUARDS

A mismatch between the number of input file and output files is more apt to be accidental than something you perform intentionally. The key point is that Human Interface commands always attempt to perform as many of your command line specifications as possible, without destroying the integrity of your files. As the operation for each listed file is successfully executed, the command displays a message that identifies the file and defines precisely what operation was performed.

When a command encounters a parameter item that is subject to ambiguous interpretation or could result in the accidental destruction of an existing file, the command usually displays a query message and prompts you to confirm or cancel the specific operation for that file. See the command descriptions in Chapter 3 for a definition of the messages.

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. You can use the commands to perform a number of highly convenient file management functions or to invoke various utility routines.

If you are a new user of the Human Interface, it is suggested that you review the command syntax and invocation considerations in Chapter 2 before reading the command descriptions provided in this chapter.

The commands described in this chapter assume that your iRMX 86 system is configured with the Human Interface commands resident in secondary storage under the :SYSTEM: directory), and any user-created commands resident in the :PROG: directory. When you enter any command (Intel-supplied or user-created) using this configuration, the Human Interface always searches for the command in the :PROG: directory first and the :SYSTEM: directory second.

The commands are presented in alphabetical sequence without regard for functional organization. A functional grouping of the commands is given in the Human Interface Command Dictionary in Table 3-1 for fast reference.

COMMAND SYNTAX SCHEMATICS

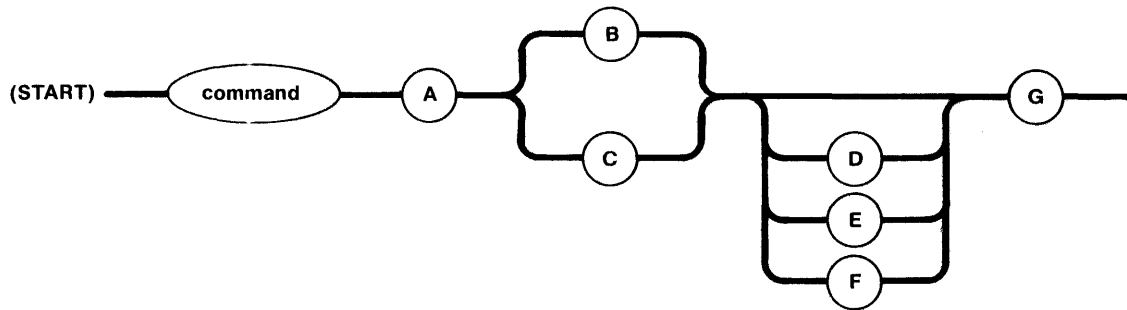
The syntax for each command described in this chapter is presented by means of a "railroad track" schematic, with syntactic elements scattered along the track. Your entrance to any given schematic is always from left to right, beginning with some command name entry.

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 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.
- Either B or C is required but not both.
- D, E, or F are all optional but only one can be selected.
- G is required.

HUMAN INTERFACE COMMANDS

Table 3-1. Human Interface Command Dictionary

Command	Synopsis	Page
	File and Volume Management Commands	
ATTACHDEVICE	Attaches a new physical device to the system and adds its logical name to the root job's object directory.	3-5
BACKUP	Copies named files to a backup volume.	3-8
COPY	Creates new data files, or copies files to other pathnames.	3-15
CREATEDIR	Creates one or more new directories.	3-18
DELETE	Deletes data files and empty directories from a volume on secondary storage.	3-22
DETACHDEVICE	Removes a physical device from system use and deletes its logical name from the root job's object directory.	3-24
DIR	Lists a directory's filenames (and optionally, file attributes).	3-25
DISKVERIFY	Verifies the data structures of named and physical volumes.	3-32
DOWNCOPY	Copies files and directories from an iRMX 86 volume mounted on a secondary storage device to an ISIS-II secondary storage device.	3-37
FORMAT	Formats an iRMX 86 volume.	3-40
RENAME	Renames files or directories.	3-45
RESTORE	Copies files from a backup volume to a named volume.	3-48
UPCOPY	Copies files and directories from an ISIS-II secondary storage device to an iRMX 86 volume mounted on a secondary storage device.	3-59

HUMAN INTERFACE COMMANDS

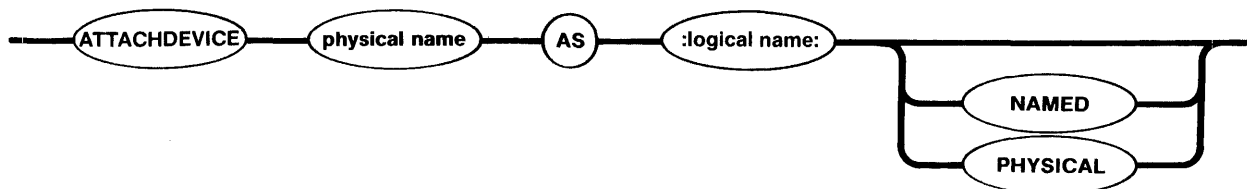
Table 3-1. Human Interface Command Dictionary (continued)

Command	Synopsis	Page
	General Utility Commands	
DATE	Sets or resets the system date, or displays the current date.	3-20
DEBUG	Transfers control to the iSBC 957A/B package to debug an iRMX 86 application program.	3-21
SUBMIT	Reads, loads, and executes a string of commands from secondary storage instead of the keyboard.	3-55
TIME	Sets or resets the system clock, or displays the current system time.	3-58

ATTACHDEVICE

This command attaches a physical device to the iRMX 86 Operating System and catalogs the device's specified logical name in the root job's object directory.

The format of the command is as follows:



INPUT PARAMETERS

- physical device** Physical device name of the device to be attached to the system. This name should be the name of one of the Basic I/O System's Device Unit Information Blocks (DUIB), as defined at system configuration time.
- AS :logical name:** Preposition plus logical name that causes the physical device to be cataloged in the object directory AS the specified logical name. After the device is attached and cataloged via the ATTACHDEVICE command, any command or program code that accesses the device must specify the logical name unless the device uses the default prefix as its logical name.
- NAMED** Specifies that the volume mounted on the device is already formatted for NAMED files. Examples of named-file volumes 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 already formatted as a single, large file. An example is a line printer. See the FORMAT command in this chapter for a further description of PHYSICAL volumes.

ATTACHDEVICE

DESCRIPTION

A physical device must be attached to the system and cataloged in the root job's object directory before it can be accessed via the logical name you assigned at the time the device was attached. Physical devices must have been declared in the Basic I/O System's Device Unit Information Block (DUIB) at configuration time before they can be attached with the ATTACHDEVICE command.

The most 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 the system. (See the DETACHDEVICE command in this chapter for a description of how to detach a device from the system without reconfiguring.)

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

physical name, attached as logical name

where "physical name" and "logical name" will be as specified in the ATTACHDEVICE command.

ERROR MESSAGES

logical name, device already attached

The specified device is attached but is not cataloged. 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, there is no DUIB defined for the device you specified. ATTACHDEVICE does not attach the device.

logical name, invalid logical name

The logical name specification is not enclosed with colons, contains unmatched colons, is longer than 12 characters, or contains invalid characters. 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.

physical device, may not be attached as a (NAMED or PHYSICAL)

The NAMED or PHYSICAL specification in the command is not allowed for that physical device; for example, defining a line printer as a NAMED volume. ATTACHDEVICE does not attach the device.

008A : E\$CONTROL, 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.

logical name, volume not formatted

ATTACHDEVICE attempted to attach a device as a named device and encountered an I/O error while searching for the volume's root directory. 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.

logical name, exception code

ATTACHDEVICE was unable to attach the specified device. This message lists the iRMX 86 exception code encountered.

BACKUP

This command saves files in 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 files beginning with the root directory of the volume.

'dd mmm yy'

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. 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.

mmm Three-character abbreviation for the month, as follows:

JAN	APR	JUL	OCT
FEB	MAY	AUG	NOV
MAR	JUN	SEP	DEC

yy Two-digit number that specifies the year.

If you omit this 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.

INPUT PARAMETERS (continued)

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
- Extension data
- User ID of the file owner
- 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 supply 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 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 all 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.x
```

where Vx.x is the version number of the utility. It then prompts you for a backup volume.

DESCRIPTION (continued)

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:

<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, enter Y to overwrite:

where backup device is the logical name of the backup device, nn is the volume number of the backup volume, and date is the date 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.

DESCRIPTION (continued)

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

pathname, SAVED

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 later restore files to a named volume with the RESTORE utility, you must supply the backup volumes in order.

ERROR MESSAGES

backup device, backup volume #nn, date, 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 code

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, the volume is write protected, 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 code, 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, error writing volume label
backup device, exception code

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 not formatted. The second line of the message indicates the iRMX 86 exception code encountered. BACKUP reprompts for a different backup volume.

ERROR MESSAGES (continued)

pathname, file does not exist

The pathname you specified as input to BACKUP does not represent an existing file or device.

backup device, invalid backup device

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

exception code, 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, invalid logical name

The logical name you specified for the backup device contains unmatched colons, is longer than 12 characters, contains invalid characters, or does not exist.

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

The backup volume you supplied was formatted as a named volume or contained some other information. 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.

keyword, too many values

You entered too many values with either the DATE or TIME parameters. The keyword portion of the message indicates the parameter that is in error.

keyword, unrecognized control

You entered one of the optional parameters of the form "keyword=value," but the keyword was not DATE, TIME, or QUERY.

ERROR MESSAGES (continued)

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 code

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.

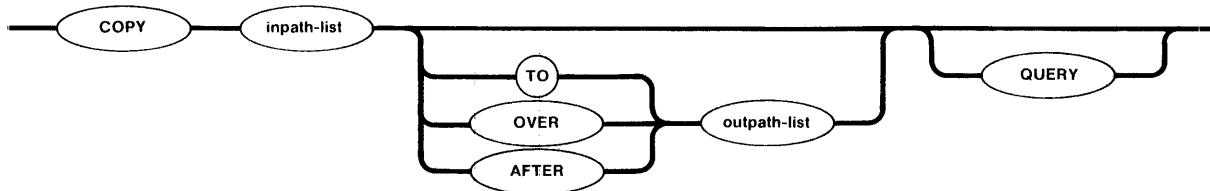
pathname, exception code

The pathname you specified as input to BACKUP is in error. This error could occur if you specify the same logical name that you specified for the backup device. It could also occur if you specify an invalid or nonexistent path component. This message displays the exception code that results from this error.

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 upon 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	Writes the listed input files to named new output files. The specified output file or files should not already exist; if they do, COPY will request permission to delete the existing files before it executes the copy operation for that file. If more input files than output files are listed, the remaining input files will be appended to the end of the last listed output file.
OVER	Writes the listed 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 listed output file name. If more input files than output files are listed, the remaining input files will be appended to the end of the last listed output file.
AFTER	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.
outpath-list	One or more pathnames for the output files. Multiple pathnames must be separated by commas. Separating blanks are optional. If the preposition and output parameter defaults are exercised in the command line, the output will go to the user's console screen (TO :CO:).

DESCRIPTION

COPY is a powerful and versatile command with a wide range of file handling applications (see Chapter 4 for examples). Implementation depends upon your selection of a preposition and your input file and output file specification in the command line. The following are some of the COPY command's features:

- Create new files (TO preposition).
- Copy over existing files or create new files (OVER preposition).
- Add data to the end of existing files (AFTER preposition).
- Copy a list of files to another list of files on a one-for-one basis.
- Concatenate two or more files into a single output file.

DESCRIPTION (continued)

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

pathname, copied TO out-pathname

pathname, copied OVER out-pathname

pathname, copied AFTER out-pathname

If you do not specify a preposition or output file, TO :CO: is the default output. The Human Interface normally expects all listed output files to be new files when the TO preposition is used; however, it is prepared to deal with existing files. If an existing output file name is encountered during a copy operation using TO, the Human Interface displays the the following message:

pathname, already exists, DELETE?

Enter Y or y if you wish to delete the existing file. The COPY command will delete the file.

Enter any other character if you do not wish to delete the existing file. The COPY command will pass over the corresponding input file without copying it, and will attempt to copy the next listed input file to its corresponding output file.

If more input files than output files are specified, the remainder of the input files will be appended to the end of the last listed output file. As each file is appended, the following message is displayed on the console screen:

pathname, copied AFTER out-file

If there are fewer input filenames than output filenames specified in the COPY command (regardless of the preposition), the output files remaining after the last valid copy operation will be ignored.

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:

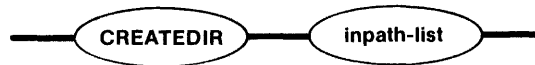
copy samp/test/a to :fl:/alpha/beta

would copy the A data file to a different volume, directory, and filename, where the new file's pathname would be :fl:/alpha/beta.

CREATEDIR

This command creates one or more iRMX 86 user directories.

The format is as follows:



INPUT PARAMETER

inpath-list	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

A created iRMX 86 directory allows all access functions; that is, you can read/write, delete, list, add, and change the contents of the directory you created with CREATEDIR.

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
```

would cause the newly-created directory GH to be nested within existing directory EF, which in turn, is nested within directory DC, and so on.

It is suggested that you use uppercase letters when you enter a new directory name in a CREATEDIR command, and use lowercase letters when you create a new data filename in a COPY command. You can then easily distinguish between directory names and filenames in a directory listing.

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 MESSAGES

directory name, invalid file type

ERROR MESSAGES (continued)

You attempted to create a directory using a data file as part of the new directory's pathname; only other directory names are allowed in the pathname for a new directory.

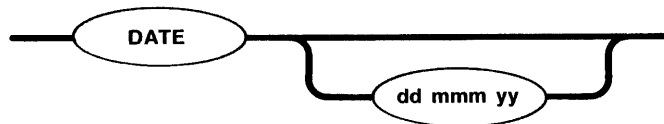
directory-name, directory already exists

The specified 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.
mmm	Three-character abbreviation for the specified month, as follows:
	JAN APR JUL OCT
	FEB MAY AUG NOV
	MAR JUN SEP DEC
yy	Two-digit number that specifies the year.

DESCRIPTION

The dd, mmm, and yy entries are separated by single blanks.

If no new date is specified in the DATE command, the current date is displayed.

If one of the date entries in the parameter string is set, all three must be; there are no default settings for individual entries within the parameter string.

If you request the system date on a non-timing system, the following message will be displayed:

00:00:00

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

ERROR MESSAGES

Errors in a date entry, such as syntax errors or a number out of range (i.e., 31 FEB 81), cause the following error message to be displayed:

illegal date

If this occurs, reenter the DATE command with the correct syntax.

DEBUG

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



INPUT PARAMETERS

- | | |
|------------------|--|
| command 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 957A/B package. You can then use the iSBC 957A/B package to single-step, display registers, and set breakpoints within the program. Refer to the appropriate iSBC 957A or iSBC 957B user's guide for a complete description of the iSBC 957A/B functions.

When DEBUG executes, the 957A/B package runs with its interrupts disabled. Therefore, the time-keeping function is also disabled, with the following consequences:

- Impacts 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.
- Impacts the ability of the Basic I/O System to keep track of the time-of-day and write its data structures to secondary storage.

The 957A/B package cannot tolerate interrupts while the single-stepping command is being used. Single-stepping will be affected if:

- Tasks are using non-zero time-out values in system calls such as RECEIVE\$UNITS and RECEIVE\$MESSAGE.
- Time-of-day is configured in the Basic I/O System.
- Non-zero update timeout values are specified in the Basic I/O System's Device Unit Information Blocks (DUIB).

The alternative to single-stepping is to use breakpoints.

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 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. If a file to be deleted is currently attached, it will be marked for deletion and deleted when the file is detached.

The following message is displayed as each file is deleted or marked for deletion:

pathname, deleted

ERROR MESSAGES

pathname, DELETE access required

You do not have permission to delete a specified file.

pathname, does not exist

The specified file was not found (e.g., a syntax error in a pathname or the file is located in some other directory). The DELETE command will attempt to delete each succeeding file specified in the filename-list after it has encountered an error in a file name.

pathname, directory not empty

Non-empty directories may not be deleted. You attempted to delete a directory that still lists filenames or other directory names.

If you still wish to delete the directory, you must first delete all its contents. For example, if you wished to delete a directory named ALPHA that contained a data file with the pathname ALPHA/BETA/SAMP, you would enter the following command:

```
delete alpha/beta/samp,alpha/beta,alpha
```

which would delete all files cataloged in ALPHA before the directory itself was deleted.

DETACHDEVICE

This command detaches the specified logical device and deletes the entry from the root job's object directory.

————— **DETACHDEVICE** ————— **:logical name:** —————

INPUT PARAMETER

:logical name: Logical name of the physical device that is to be deleted from the root job's object directory.

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. For a description of formatted volumes (NAMED or PHYSICAL), see the FORMAT command description in this chapter.

When the device is detached and its logical name has been deleted from the root job's object directory, the DETACHDEVICE command will display 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 MESSAGE

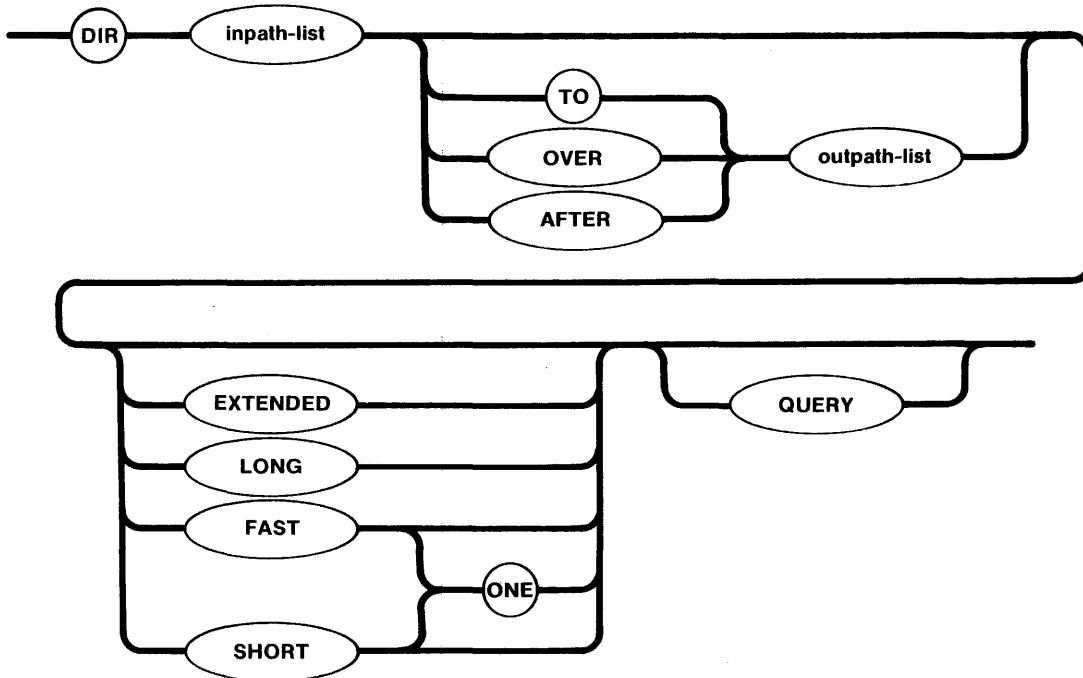
illegal logical name

Either there is a syntax error in the logical name specification or the logical name does not exist in the root job's object directory.

DIR

This command lists the names and attributes of files contained in a given directory, including data filenames and directory names.

The format of the command is as follows:



INPUT PARAMETERS

inpath-list One or more pathnames of the directories to be listed. 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.

EXTENDED Lists all available information for each data file or directory file in the directory. The first line for each file will be the same as for the LONG form. The second line will contain the last access date, creation date, and the accessor list. The listing will be in a double-column format (see Figure 3-1 at the end of this command description).

LONG Lists file information in a one-line format (see Figure 3-2 at the end of this command description).

INPUT PARAMETERS (continued)

- FAST** Lists only the filenames and directory names in the directory. The output format will be five columns of filenames unless you also specify the ONE parameter (see Figure 3-3 at the end of this command description). If no listing format is specified, FAST is the default.
- SHORT** Lists the file information in a two-column format (see Figure 3-4 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.
- 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 no TO/OVER/AFTER preposition is specified, 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.
- out-pathname** Pathname of the file to receive the directory listing. If the parameter is omitted, the default destination is the user's console screen (TO :CO:).

DESCRIPTION

The amount of information listed for each file depends upon what listing format you specify (EXTENDED, FAST, LONG, or SHORT) in the DIR command. An example of each type of listing format is provided at the end of the DIR command description in Figures 3-1 through 3-4 respectively. An explanation of the illustrated headings is provided in Table 3-2 following the figures.

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
```

would display a listing of the user directory in the EXTENDED format. Note that the default directory is a configuration option.

ERROR MESSAGES

```
pathname, is not a directory
```

A pathname exists but is not a directory.

```
pathname, directory does not exist
```

The pathname does not exist, either as a directory or as a data file.

```
pathname, directory LIST access required
```

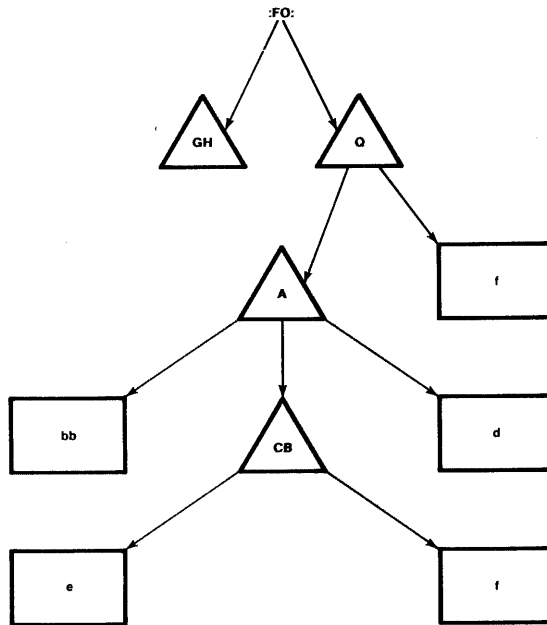
You do not have list access to the directory.

DIR COMMAND EXAMPLES

The examples that follow show how a directory's files are listed when you use your configured system's 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:

DIR COMMAND EXAMPLES (continued)



Example 1:

If your default prefix was :FO:/Q, then the following files would be listed in response to the DIR pathname entry examples in the following "Pathname" column:

<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:

If your default prefix was :FO:/Q/A, then the following files would be listed in response to the DIR pathname entry examples in the following "Pathname" column:

<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

DIR LISTING FORMATS

Figures 3-1, 3-2, 3-3, and 3-4 show output examples for EXTENDED, FAST, LONG, and SHORT listing formats respectively. Table 3-2 defines the displayed column headings.

(column scale only)

1 2 3 4 5 6 7

123456789012345678901234567890123456789012345678901234567890123456789

11 MAR 80 06:30:30
 DIRECTORY OF sys ON myvol

NAME	AT	ACC	BLKS	LENGTH	VOL FIL		OWNER	LAST MOD
ed		DR	200	30185	16	3	Beck	20 NOV 79
				CREATION:	20	APR	78	ACCESSORS
				LAST ACC:	25	NOV	79	Engineers
							Techs	R
								U
idisk	DR	DLAC	5	39	16	1	WORLD	12 DEC 79
				CREATION:	15	NOV	78	ACCESSORS
				LAST ACC:	10	JAN	80	ACC
submit\$plm\$ab	MA	DRAU	11	1057	24	2	BACKWORDPLMCOM	16 JUN 79
				CREATION:	20	APR	78	ACCESSORS
				LAST ACC:	20	JAN	80	PYE-WACKET
							TOGAN	RAU
							longlongnames	R
								D
								U
CHAOTICGOOD		U	123456789	1234567890	12345	123	Chopin	01 DEC 79
							ACCESSORS	ACC
							Clerics	RA
							MAGIC-USERS	U
							Thieves	D
							FIGHTERS	DRAU
LAWFULEVIL		D	73	9081	24	15	saveyourhat	04 JAN 80
				CREATION:	15	NOV	79	ACCESSORS
				LAST ACC:	05	MAR	80	WORLD
								RAU
				5 FILES				1839 BLOCKS
								1200453 BYTES

Figure 3-1. EXTENDED Directory Listing Example

03/11/80 04:25:40
 DIRECTORY OF alpha ON mvol
 fname1 fname2 fname3 fname4 fname5
 fname6 fname7 fname8 fname9 fname10
 fname11 . . .
 .
 .
 .

Figure 3-2. FAST Directory Listing Example

DIR LISTING FORMATS (continued)

(column scale only)

1 2 3 4 5 6 7

123456789012345678901234567890123456789012345678901234567890123456789

11 JAN 80 06:30:30
 DIRECTORY OF sys ON myvol

NAME	AT	ACC	BLKS	LENGTH	GRAN		OWNER	LAST MOD
ed		DR	200	30185	16	3	BECK	20 NOV 79
idisk	DR	DLAC	5	39	16	1	WORLD	12 DEC 79
LEMONADEIT	MA	D	105	13074	64	2	malagi	16 MAR 77
credit		DR	263	32967	128	6	WORLD	17 NOV 79
SUBMITAGAINPLM	MA	DRAU	11	1057	24	2	BACKWORDPLMCOM	16 JUN 79
type	DR	LA	4	366	16	1	PASCAL	15 DEC 79
CHAOTICGOOD		U	123456789	1234567890	12345	123	CHOPIN	01 DEC 79
LAWFULEVIL		D	73	9081	24	15	saveyourpet	04 JAN 80
8 FILES			1839 BLOCKS	1200453 BYTES				

Figure 3-3. LONG Directory Listing Example

(column scale only)

1 2 3 4 5 6 7

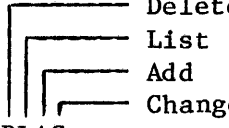
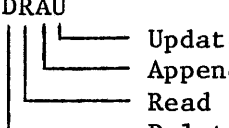
123456789012345678901234567890123456789012345678901234567890123456789

03/11/80 04:25:50
 DIRECTORY OF sys ON myvol

NAME	AT	ACC	BLKS	LENGTH	NAME	AT	ACC	BLKS	LENGTH
append		R	40	1425	attrib		DRAU	38	4682
COPY	MA	DRAU	65	8042	CREDIT.HAZ		R	263	33017
dcopy		DRAU	62	7718	DELETE		A	37	4506
REFERENCE	DR	L	5	10	DATA	DR	DLAC	1	4
DUMP		D	22	2568	ED		DR	200	30185
idisk	DR	DLAC	5	39					
LEMONADEIT	MA	D	123456789	1234567890					
CREDIT		DR	263	32967	RENAME		AU	21	2487
submit\$plm	MA	DRAU	11	057	TYPE	DR	LA	4	366
CHAOTICGOOD		U	13293	1151640	lawfulevil		D	73	9081
18 FILES			1839 BLOCKS	1200453 BYTES					

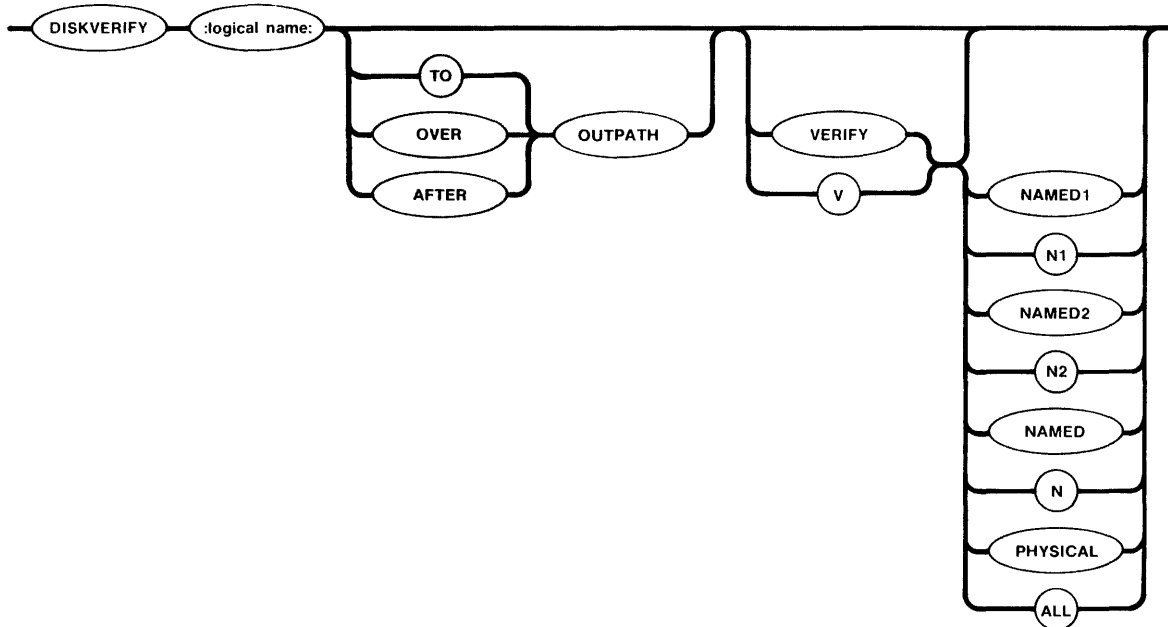
Figure 3-4. SHORT Directory Listing Example

Table 3-2. Directory Listing Headings

Heading	Meaning
NAME:	14-character file NAME
AT:	File ATtribute, where: DR = Directory (if the ATtribute field is blank, the file is a data file)
ACC:	File ACCess rights, where: <div style="display: flex; align-items: center; margin-left: 40px;"> <div style="margin-right: 10px;">Directories:</div> <div style="margin-right: 10px;">DLAC</div> <div style="margin-right: 10px;">  </div> </div> <div style="display: flex; align-items: center; margin-left: 40px;"> <div style="margin-right: 10px;">Other Files:</div> <div style="margin-right: 10px;">DRAU</div> <div style="margin-right: 10px;">  </div> </div>
BLKS:	Nine-digit number (five digits on SHORT listing) giving the volume-granularity units allocated to the file. On the SHORT form, if the number of digits exceeds five, the file is displayed in the nine-digit form (see the LEMONADEIT file in Figure 3-4).
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 3-4).
VOL:	Five-digit number giving the volume granularity in bytes.
FIL:	Three-digit number giving the granularity of the file in volume-granularity units.
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.
ACCESSORS:	Heading for list of 14-character accessor names.
ACC:	Heading for access rights of file accessors. The format of this field is identical to ACC above.

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:



INPUT PARAMETERS

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

INPUT PARAMETERS (continued)

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 heirarchy. 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.
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.
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.
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.
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 the PHYSICAL verification function.

OUTPUT PARAMETERS

TO	Copies the output from the disk verification utility to the specified 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.

OUTPUT PARAMETERS (continued)

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.x
```

where Vx.x is the version number of the utility. If you specify the VERIFY or V parameter in the DISKVERIFY command, the utility performs a verification of the volume and copies the verification information 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 verification output. After generating the verification output, the utility returns control to the Human Interface, which prompts you for more Human Interface commands. The following is an example of such a DISKVERIFY command:

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

'NAMED2' VERIFICATION

BIT MAPS O.K.
```

However, if you omit the VERIFY (or V) parameter 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

logical name, 0045 : E\$LOG_NAME_NEXIST

You specified a nonexistent logical name in either the :logical name: parameter or the outpath parameter.

8042 : E\$NOT_CONNECTION

You attempted to direct output to a file on the volume being verified.

command line 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.

logical name, illegal logical name

The logical name you specified was not surrounded by colons (:).

not a named disk

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

verify-function argument error

The VERIFY option you specified is not valid.

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.

DISKVERIFY

EXAMPLE

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

-DISKVERIFY :F1: VERIFY ALL

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

'NAMED1' VERIFICATION

'NAMED2' VERIFICATION

BIT MAPS O.K.

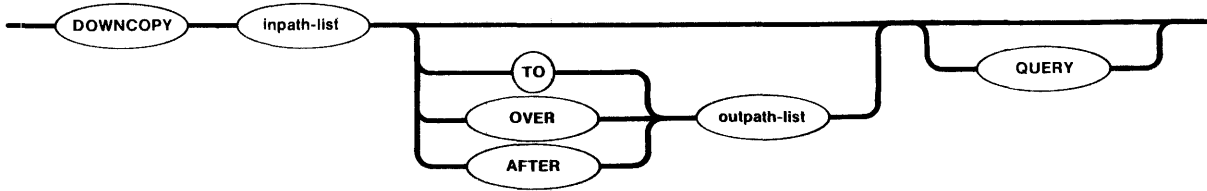
'PHYSICAL' VERIFICATION

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 957A/B 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 out-pathname?

pathname, copy down OVER out-pathname?

pathname, copy down AFTER out-pathname?

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.

DOWNCOPY

OUTPUT PARAMETERS

TO	<p>Reads iRMX 86 files and copies them TO new ISIS-II files in the listed sequence. The specified output files should not already exist in the ISIS-II directory when the TO parameter is used. If a named output file does exist, DOWNCOPY will display the following message:</p> <p style="padding-left: 40px;">filename, already exists, delete?</p> <p>Enter a Y or y if you wish to delete the existing file. Enter any other character if you do not wish the existing file to be deleted.</p> <p>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 will be appended to the end of the last listed ISIS-II file.</p>
OVER	<p>Reads the listed iRMX 86 input files and copies them OVER the existing ISIS-II destination files in the listed sequence. If more input files than output files are listed, the remaining input files will be appended to the end of the last listed ISIS-II file.</p>
AFTER	<p>Reads the listed iRMX 86 input files and copies them AFTER the end of data on the existing ISIS-II destination files in the listed sequence.</p>
outfile-list	<p>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 will go to the ISIS-II console screen.</p>

DESCRIPTION

The DOWNCOPY command cannot be used to copy directories from an iRMX 86 system to an ISIS-II system; only files can be copied.

Before you enter a DOWNCOPY command on the iRMX 86 console keyboard, you must have your target system connected to a development system with the 957A/B package, and the package must be running. The ISIS-II copies of the files will have all ISIS-II file attributes turned off.

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:), as appropriate:

DESCRIPTION (continued)

pathname, copied down TO out-filename

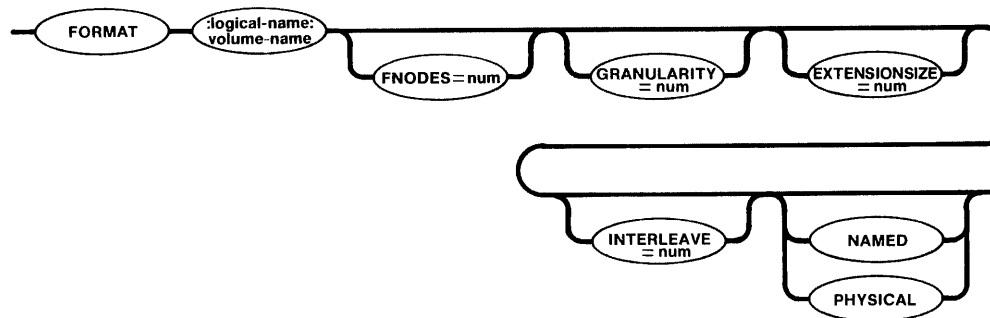
pathname, copied down OVER out-filename

pathname, copied down AFTER out-filename

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:



INPUT PARAMETERS

- :logical-name:** Logical name of the physical device-unit to be formatted. The logical name must be preceded and followed by colons without embedded blanks between the logical name and volume name.
- volume-name** Six-character, alphanumeric ASCII name, without embedded blanks, to be assigned to the volume. (See the definition for a "volume" in Chapter 1.)
- FNODES=num** Defines the maximum decimal number of files that may be created on a NAMED volume. (This parameter is not meaningful when formatting a PHYSICAL volume and will be ignored if specified for such volumes.) The range is 7 through 32,767 fnodes, although the maximum number of fnodes you can define depends on the settings of the GRANULARITY and EXTENSIONSIZE parameters (as explained in the "Description" portion of this command write-up). If not specified, the default is 50 fnodes.
- 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 will be ignored if specified for such volumes.) The specified decimal number is placed in the header of the volume and becomes the default file granularity when a file is created on the volume.

INPUT PARAMETERS

GRANULARITY=num (continued)

The range is 1 through 65,535 (decimal) bytes, although the maximum allowable volume granularity depends on the settings of the FNODES 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.

EXTENSIONSIZE=num Size, in bytes, of the extension data portion of each file descriptor node (fnode). (This parameter is not meaningful for PHYSICAL volumes, and will be ignored if specified for such volumes.) The range is 0 through 65,448 (decimal), although the maximum allowable extension size depends on the settings of the FNODES 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. If not specified, the default value is 5, which is the optimum interleave factor for an iSBC 204 bootstrap load. 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 type of files 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 FNODES parameters are not meaningful when PHYSICAL is specified for the volume. If neither NAMED nor PHYSICAL is specified, the volume is formatted for the type of files 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 list the root directory for the volume, and the name will appear in the directory's heading. Although the Human Interface uses the volume name in its own internal processing when you access the volume, you do not need to specify the volume name in any subsequent command after the volume is formatted; only the logical name of the secondary storage device on which the volume is currently mounted needs to be specified.

The number of fnodes on a volume defines the number of files that can exist on the volume. You can specify this number with the FNODES 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 in order to determine the location of the file on the volume.

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 SYSTEM PROGRAMMER'S 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 65,448 bytes, the Human Interface requires all fnodes to have at least 2 bytes of extension data.

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 128 bytes of permanent storage to each new file you create on that volume, regardless of whether or not a file requires a full 128 bytes. If the size of a file exceeds 128 bytes, the I/O System will allocate still another full block of 128 bytes, and so on, until the volume is full.

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

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

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

DESCRIPTION (continued)

As stated previously, the interleave factor applies to volumes formatted either for NAMED or PHYSICAL files. 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 will read every fifth sector on the diskette, starting with an index of 1 (other, hard disk systems may be different, depending on your 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 previously accessed sector has been processed.

The FORMAT command displays the following message when volume formatting is completed:

```

    volume (vol-name) will be formatted as a NAMED/PHYSICAL volume
      granularity = number
      interleave = number
      numberfnodes = number
      extensionsize = number

```

where:

vol-name	The volume name specified in the FORMAT command.
NAMED/PHYSICAL	Either NAMED or PHYSICAL will be displayed in the message, depending on the command specification.
number	Default or specifically defined in the command.

ERROR MESSAGES

If a device cannot be detached for formatting, the following message is displayed on the user console:

```

    logical-name, can't detach device

```

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.

If the device cannot be attached for formatting, or it cannot be re-attached (e.g., restored to its original configuration prior to formatting) after formatting takes place, the following message is displayed on the user console:

```

    device-name, can't attach device

```

ERROR MESSAGES (continued)

The following error message is displayed if you attempt to format something that is not a physical device:

logical-name, is not a device connection

The following error message is displayed if you specify a volume name containing more than six ASCII characters or if you specify a logical device name:

vol-name, illegal name

The following error message is displayed if you specify an out-of-range number for any of the FNODES, GRANULARITY, EXTENSIONSIZE, or INTERLEAVE parameters:

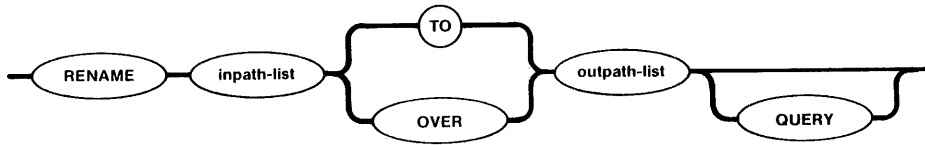
number, illegal number

The following message is displayed if the values you specify for fnode size, granularity, and extension data size cause the formula listed in the "Description" section to exceed its limit.

vol-name, fnode file size exceeds 65535 volume blocks

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 the following message:

oldname, RENAME?

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, in fact, a new pathname does already exist, RENAME displays the following warning message when the pre-existing pathname is encountered:

pathname, already exists, DELETE?

RENAME

OUTPUT PARAMETERS

- TO (continued) Enter a "Y" or "y" if you wish the pre-existing pathname and its contents to be written over by the new name specification. The pre-existing pathname and its contents will be deleted.
- Enter any other character if you do not wish the pre-existing file to be deleted. Renaming of the specified file will not take place and the RENAME command will attempt to rename the next pathname in the list sequence.
- OVER Changes each old pathname in a list to the corresponding new pathname, even if the new pathname already exists. The old pathname is deleted from secondary storage. 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 a RENAME command is executed, it releases the pathnames in the listed input files for new uses without having to perform any further operation on the files.

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/sampl over BETA/test1      allowed
```

CAUTION

Note that changing the name of a directory also changes the path of all files listed under 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

There must be a one-for-one correspondence between the oldname and newname lists in the RENAME command. A missing element in either list causes RENAME to display the following message:

unmatched path name lists

If your system is configured with user-designed access limitations, you must have at least delete access to old pathnames and add-entry access to the destination directory to use the RENAME command.

If you are not allowed delete access on your system, the following message is displayed when you attempt to use the OVER preposition in a RENAME command:

old-pathname, DELETE access required

If you are not allowed add-entry access on your system, the following message is displayed when you attempt to use the TO preposition in a RENAME command:

new-pathname, directory ADD ENTRY access required

If the RENAME command encounters an error in the renaming of a file, it will attempt to continue renaming each succeeding file in sequence.

Use of the AFTER preposition is not valid for the RENAME command, and an attempt to use it causes the following message to be displayed:

AFTER preposition, TO or OVER preposition expected

Note that the RENAME command is the only Human Interface file handling command that 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 (e.g., from one diskette to another). An attempt to do so causes the following error message:

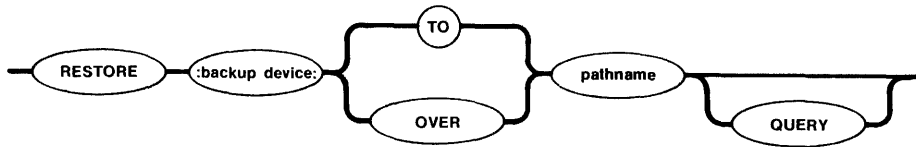
0005: E\$CONTEXT

Use the COPY command or a combination of COPY and DELETE commands if you wish to rename files or move data across volume boundaries.

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.

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. However, if a file being restored already exists on the named volume, RESTORE prompts for permission to restore the 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.

When RESTORE copies files, it copies only those files for which you are the owner. For these files, it 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.

DESCRIPTION (continued)

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. In order for RESTORE to create new files on a named volume, you must have add entry access to directories on that volume. In order 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.x
```

where Vx.x 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 continues prompting you until you supply the correct backup volume.

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

```
pathname, RESTORED
```

However, if a file with the same pathname already exists during a restore operation using the TO preposition, RESTORE displays the following message:

```
pathname, already exists, DELETE?
```

DESCRIPTION (continued)

Enter one of the following in response to the query:

<u>Entry</u>	<u>Action</u>
Y or y	Delete the file and replace it with the one from the backup volume.
E or e	Exit from the RESTORE command.
R or r	Delete the file, replace it with the one from the backup volume, and continue restoring files without further queries.
Any other character	Do not restore the file; go on to the next file.

ERROR MESSAGES

pathname, ADD ENTRY or UPDATE access required

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 code

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, the volume is write protected, 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.

ERROR MESSAGES (continued)

pathname, DELETE access required

RESTORE could not restore a file because you did not have delete access to the file. RESTORE continues with the next file.

pathname, exception code, 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 code, 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 code

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 code, 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 completely 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. RESTORE restores as much of the extension data as possible. 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.

ERROR MESSAGES (continued)

pathname, file does not exist

The pathname you specified as input to RESTORE does not represent an existing file or device.

pathname, file 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.

backup device, invalid logical name

The logical name you specified for the backup device contains unmatched colons, is longer than 12 characters, contains invalid characters, or does not exist.

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.

backup device, not a valid backup device

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

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.

keyword, too many values

You specified too many values after the TO or OVER parameter.

RESTORE

ERROR MESSAGES (continued)

keyword, unrecognized control

You entered an invalid optional parameter. The keyword portion of the message indicates the parameter that is in error.

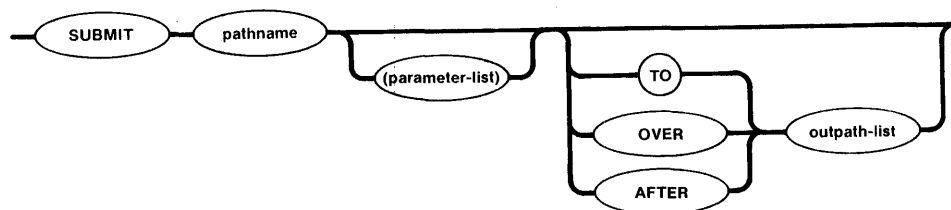
pathname, exception code

The pathname you specified as input to RESTORE is in error. This error could occur if you specify an invalid or nonexistent path component. This message displays the exception code that results from this error.

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 listed output file already exists, the SUBMIT command will display the following message:
----	--

pathname, already exists DELETE?

Enter a Y or y 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.

OUTPUT PARAMETERS (continued)

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.
outpath-list	Pathnames of one or more files 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 keyboard 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. When one nested SUBMIT file completes execution, it returns control to the next higher level of SUBMIT file.

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 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

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:PROGRAM, 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.

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

```
-SUBMIT :F1:PROGRAM (:F1:, PROG, FILE1)
```

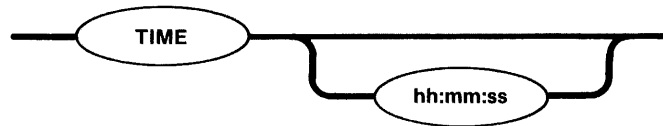
The command sequence created and executed by SUBMIT is shown as it would be echoed on the console output device.

```
-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 :F1:PROGRAM
-
```

TIME

This command sets the system clock. If no new time is entered, the TIME command causes the current system time to be displayed.

The format is as follows:



INPUT PARAMETERS

hh:	Hours specified as 0 through 24.
mm:	Minutes specified as 0 through 59.
ss	Seconds specified as 0 through 59.

DESCRIPTION

If one of the time entries in the parameter string is set, all three must be; there are no default settings for individual items in the parameter string.

If you request the time-of-day and the system clock has not been set, the TIME command displays the following message:

```
00:00:00
```

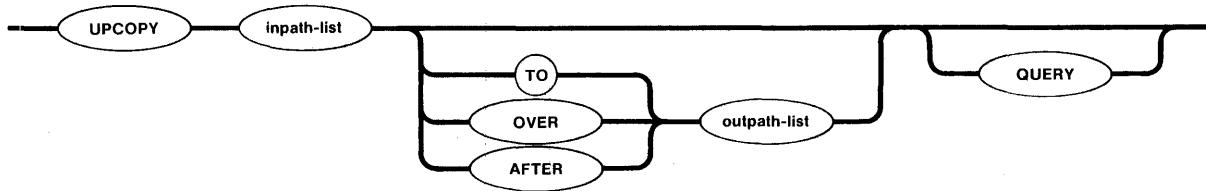
See also the DATE command in this chapter if you wish to set the date in conjunction with the system clock.

An invalid time or an out-of-range entry for the TIME command causes the following error message to be displayed:

```
illegal time
```

UPCOPY

This command copies files from a volume on ISIS-II secondary storage to a volume on iRMX 86 secondary storage via the iSBC 957A/B 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:

filename, copy up TO out-pathname?

filename, copy up OVER out-pathname?

filename, 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** Copies the ISIS-II file or files TO a new iRMX 86 file or files in the listed sequence. The output file or files should not already exist when the TO preposition is used. 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 inputfiles will be appended to the end of the last listed output file.
- OVER** 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.
- AFTER** 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.
- outpath-list** 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.

DESCRIPTION

Before you enter an UPCOPY command on the iRMX 86 console keyboard, you must have your target system connected to a development system with the 957A/B package and the package must be running. The iRMX 86 copies of the files will have WORLD access; that is, all iRMX 86 system users can perform read, write, and delete operations on the files without restriction.

As each ISIS-II file in the input list is copied, the Human Interface will display one of the following messages on the iRMX 86 console screen, as appropriate:

- filename, copied up TO out-pathname
- filename, copied up OVER out-pathname
- filename, copied up AFTER out-pathname

CHAPTER 4. FILE HANDLING EXAMPLES

This chapter shows you how to use the Human Interface file management commands at the user console. The primary intent is to introduce you to basic file manipulation techniques by presenting a series of examples that illustrate typical command entries.

Those of you who will be using the Human Interface system calls to create your own commands can also benefit by studying the examples. Hands-on experience with the file handling commands will provide you with an improved understanding of the interaction between the console user and Human Interface services during program execution. You will then have a better insight as to why and how some of these same elements should be included in your own custom commands.

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 parenthesis in the examples to indicate that such entries are not echoed on the console screen as they are entered. Do not actually enclose control key entries with parentheses.

HOW TO BEGIN A CONSOLE SESSION

You can begin an interactive dialog with the Human Interface after a sign-on message is displayed on your console screen. Although the sign-on message is a system configuration option, the message supplied with Human Interface systems is as follows:

```
iRMX 86 HI Vx.x: user = WORLD
-
```

FILE HANDLING EXAMPLES

where the message tells you the Human Interface is running, and 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
```

HOW TO CREATE A SIMPLE DATA FILE

The COPY command is the command used 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 the default user's directory configured for your system, which means you do not have to specify a directory name as a preceding component in the file's pathname. 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, which will be listed under your system's default user directory. You will not be prompted for the data lines; 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.

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, all characters entered after you press the RETURN key and before you enter a CTRL/z would not be written to the file. For example:

```
-copy :ci: to alpha  
cccc  
dddd (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  
cccc  
dddd  
  
:ci: copied TO alpha  
-
```


FILE HANDLING EXAMPLES

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, DELETE?
```

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, DELETE? y
```

the COPY command deletes the data within the existing file and waits for you to enter new data under that pathname.

If your response to the query is:

```
alpha, already exists, DELETE? 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 DUPLICATE 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 Human Interface provides the following file copy services:

- Copy files on a one-for-one basis.
- List (display) files on the console screen.
- Create multiple copies of the same file.
- Copy data from multiple files to a new or existing file.
- Replace data in one file with data from another file.
- Add data from one or more files to the end of the data in another file.
- Combine one-for-one file copying with file concatenation in a single COPY command.

The examples that follow show you how to implement these various options and also call your attention to certain file handling considerations when using the COPY command.

FILE HANDLING 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 ALPHA and BETA to files GAMMA and DELTA respectively. Enter the following command:

```
-copy alpha,beta to gamma,delta  
alpha copied TO gamma  
beta copied TO delta  
-
```

There is a rule to remember about copying lists of files. Although you can list more input files than output files (for file concatenation, which is described later), you cannot successfully list more output files than input files in a command line because the command would not know how to handle the remaining output files. The COPY command will copy input files to listed output files until the input list is exhausted, but no operation will be performed for the extra output files.

HOW TO DISPLAY FILES

When you perform a number of file manipulations during a single session, it is occasionally advisable to check a file's contents by listing it on the the screen before proceeding further. Assume you wish to display the contents of a file named ALPHA that is listed under the user default directory. Simply enter the command:

```
-copy alpha  
aaaaa  
aaaaa  
  
alpha copied TO :CO:  
-
```

This COPY command example used the default preposition (TO), and default output file (:CO:), which is the console screen. However, remember that one of these defaults cannot be used in a command line without using the other. For example, the commands:

```
-copy alpha to  
8MAD, 8004: E*PARAM  
-  
or  
  
-copy alpha :co:  
:CO:, unrecognized control  
-
```

obviously give poor results.

FILE HANDLING EXAMPLES

You can halt the scrolling of a displayed list to examine the data more closely. Press the following CTRL function keys to control scrolling of the listed output:

CTRL/s	Stops the data from being scrolled off the screen until you press a CNTRL/q or CTRL/c.
CTRL/q	Resumes scrolling of listed data until the end-of-file is reached or you enter a CTRL/c.
CTRL/c	Cancel 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. The most practical way to modify a file is to create a new file and then replace the contents of the old file with the new data. Although this operation can be done by using the RENAME command, for now we'll perform it by exploring the use of 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. 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 to list the file shows that the old file contents have been successfully replaced. We 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 below:

FILE HANDLING EXAMPLES

```
-copy temp to alpha  
alpha, already exists, DELETE? 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 more input files than output files; or 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.

FILE HANDLING EXAMPLES

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:  
-
```

You can also use the OVER preposition in conjunction with file concatenation. For example:

```
-copy alpha,beta,gamma over delta, zeta  
alpha copied OVER delta  
beta copied OVER zeta  
gamma copied AFTER zeta  
-
```

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
```

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:

1. There is a syntax error in the spelling of the KEY file.
2. The file does not exist.
3. 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 user directory is simply another kind of file under which you assign and maintain other files or directories. A directory file is distinguished from a data file or program 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 files and other directories that you subsequently assign to it. Directories provide you with a convenient and efficient technique for organizing large numbers of files into logical groupings. Creating your own user directories will aid you in two ways:

1. It's easier to keep track of individual files when you're maintaining large numbers of files on the system.
2. It reduces the possibility of accidental destruction of files, either by yourself or other system users.

FILE HANDLING EXAMPLES

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 have used the default user directory configured for your system to create and access files. 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 the files themselves are created. (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 have lexical meaning; that is, the names 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  
-
```

You probably noticed that we entered the directory pathnames in capital letters in the above example. It is suggested that you also capitalize all directory pathname entries in a CREATEDIR command when you create new directories, 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 file pathnames and directory pathnames.

Once the directories or files are created, you can enter their pathnames in either lowercase or uppercase characters in subsequent commands; the Human Interface commands make no distinction in interpretation.

FILE HANDLING EXAMPLES

HOW TO REFERENCE 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 you have 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 (without embedded blanks) separate the individual hierarchical components of the pathname. The pathname for the TEST data file would be NUTEST/SAMP/TEST.

HOW TO ADD NEW ENTRIES TO A DIRECTORY

Previous data file examples in this chapter used the default user directory (as configured for your system) for all file creation and access. Consequently, each time we created a new file or accessed an existing one, we only needed to enter the filename for the file; the directory name as the first component of a file's pathname did not have to be specified in a command. However, whenever you wish to create a new data file to be assigned to a specific directory, you must precede the filename by the directory name and separate the two names with a slash (/) in the COPY command, as described in the previous subsection.

For example, assume you wish to create files named SAMP1 and SAMP2 and assign them to the MYTEST 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. For example, assume you wanted to delete files SAMP1 and SAMP2 from the MYTEST directory and entered the following command:

```
-delete mytest/samp1,samp2  
mytest/samp1, deleted  
samp2, does not exist  
-
```


FILE HANDLING EXAMPLES

The Human Interface issued the "does not exist" message for SAMP2 because it looked for the file in the default user 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.

HOW TO CREATE A DIRECTORY WITHIN A DIRECTORY

No doubt you have already realized that since you can create new directories within the default user directory, you can also create other directories within a given path of directory names in an ever-descending hierarchy. This process exactly duplicates the classification of topics and subtopics within a standard office filing system.

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 pathname for the URTEST directory, and the "MYTEST/URTEST/NOMOR" sequence is the pathname for the NOMOR file. The entire pathname must be specified for directories or files in any subsequent file handling. 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:  
-
```

FILE HANDLING 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 data files by specifying a directory pathname in a COPY command. However, you should not use the COPY command because COPY will list the directory as though it were a data file. For example, if we used the COPY command to list the MYTEST directory on the screen:

```
-copy mytest
alphabetagammaurtest copied to :CO:
-
```

the resulting output is almost unreadable. Instead, use the DIR command to list the directory's catalog of data files and/or directories as follows:

```
-dir mytest

01 JAN 78 00:00:00
  DIRECTORY OF MYTEST ON VOLUME disk2
alpha          beta
  gamma        URTEST
-
```

In this example, you 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.

FILE HANDLING 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 the users default directory to the existing directory MYTEST, and file DELTA from the user 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 decided to move file ALPHA back to the default user 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 filename. 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 wished to delete directory MYTEST and did not realize it contained a data file named ALPHA and a directory named DED that itself contained a data file named LIV. You entered the following command:

```
-delete mytest  
mytest, directory not empty  
-
```

FILE HANDLING EXAMPLES

It would probably pay to list the MYTEST directory by using the DIR command at this point. You now have several options: use the RENAME command to move any files to to be saved to a different directory on the same volume, or 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 to the MYTEST directory as the last item to be deleted.

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, renaming of files and directories will be described 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 would be automatically deleted from the system when the RENAME command was executed. You can also rename lists of files to new pathnames, and 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 intended.

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  
-
```

FILE HANDLING EXAMPLES

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 console:

```
-rename alpha,beta to tom  
alpha renamed to tom  
un-matched path name lists,  
-
```

Similarly, specifying fewer old pathnames than new names would cause the following exchange:

```
-rename alpha to beta,tom  
alpha renamed to beta  
-
```

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 data file from the system 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 from the system 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  
-
```

Now, let's see what happened to the contents of the file previously named ABLE but now named BAKER:

```
-copy baker  
aaaaa
```

```
baker copied TO :CO:  
--
```

FILE HANDLING EXAMPLES

The previous contents of BAKER have been deleted from the system, 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, DELETE? y
alpha renamed TO gamma
delta, already exists, DELETE? 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 wish to rename it with a new pathanme 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 cautious about renaming directories! That last message tells you 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, either the directories would have to be renamed to their original names or the programs would have to be modified.

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 in a different drive. The restriction against using RENAME across volume boundaries is intended for the protection of files against accidental deletion.

FILE HANDLING EXAMPLES

You access a different volume by entering the logical device name (for 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 on your system's default volume (:f0:). 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 (for instance) 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 create 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:. Further assume you are going to name the volume NEWVOL to meet the Human Interface requirement for a six-character volume name.

Enter the following command:

```
-format :fl:NEWVOL
volume (NEWVOL) will be formatted as a NAMED volume
  granularity = 128
  interleave = 5
  numberfnodes = 50
volume formatted
-
```

This formatting example exercised all the default options. Only a six-character volume name is a required parameter. Since all your accesses to the volume will be through the logical name for the drive on which the volume is mounted, the question naturally arises as to why a volume name is required. There are two reasons:

1. 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 diskette.
2. 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 and FNODES 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:

- The default NAMED parameter 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.

FILE HANDLING 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 will automatically allocate 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 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 FNODES and GRANULARITY options are ignored in FORMAT commands that specify a PHYSICAL file format for the volume.

- The default FNODES parameter specifies that you wish create a maximum of 50 files on the volume, including directory files. Although the actual number of fnodes you can specify is 7 through 32,767, at a practical level one of your determining factors will be the incremental file size you specify in the GRANULARITY parameter.

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.
2. Enter a DIR command for the root directory of the new diskette to force physical access. The root directory "name" will actually be the prefix (logical device name) for the drive on which the diskette is mounted. For example:

```
-dir :fl:
```

The following exception message will be displayed:

```
E$IO
```

```
-
```

FILE HANDLING EXAMPLES

3. Ignore the error message and begin entering Human Interface commands that access the volume. If your system is configured with a dual-density disk controller, use the diskette changing procedures described in the IRMX 86 INSTALLATION GUIDE.

HOW TO USE A SUBMIT FILE

A submit file gives you the ability to load and execute an entire series of commands by entering a single SUBMIT command. Whenever you have a group of commands that you must execute in sequence and on a routine basis, using a submit file will save you time, keystrokes, and keyboard entry errors.

Before you create a submit file, however, you should be aware of two restrictions about the commands that are loaded and executed from such a file:

1. There is no way to change parameters in any given command between submit file executions. If you wish to change any parameter specification in a command line, you must recreate the entire submit file.
2. You cannot change the execution sequence for commands without recreating the file.

Now, assume you have three user-created application programs whose command names are the pathnames :F1:/STATS/PASS1, :F1:/STATS/PASS2, and :F1:/StATS/PASS3 respectively. The pathnames tell you that the volume in which they are located is mounted on the drive with the logical name :F1:, all three programs are located in the STATS directory, and the program names are PASS1, PASS2, and PASS3.

- PASS1 reads data from an update file whose pathname is always :F1:/NEWDATA, processes it, and writes it over an existing output file whose name is always :F1:/FETCH.
- PASS2 reads data from :F1:/FETCH, processes it, and writes the output after the end of data on an existing file named :F1:/CARRY.
- PASS3 reads the data from :F1:/CARRY, processes it, and writes the output over an existing file whose pathname is :F1:/TOTAL
- The Human Interface COPY command reads :F1:/TOTAL and writes it to the :CO: device (console screen). You could have PASS3 write directly to :CO: but you would then have no way to recopy the file.

Assume you have created your submit file with the pathname MYTEST/GETIT and you now wish to create the NEWDATA file from :ci: and execute the submit file. Enter the following commands:

FILE HANDLING EXAMPLES

-copy :ci: to :fl:/newdata
aaaaaa
bbbbbb
cccccc
(CTRL/z)
-SUBMIT mytest/getit

The Human Interface will read the list of commands from the GETIT file as though you were entering them from the keyboard, follow the pathnames to the stored programs, load, and execute each program in sequence.

CHAPTER 5. CREATING NEW COMMANDS

The Human Interface system calls provide a convenient and consistent method for handling parameter parsing and processing in user-designed commands. You implement parameter processing by using appropriate system calls when you write the application program.

After you have assembled or compiled, linked, and then stored your new command as an object module in secondary storage, a console operator can load the program file by typing its command name (pathname of the program file), plus any desired or required parameters on the console keyboard.

GENERAL COMMAND LINE STRUCTURE

The general structure for a command line is as follows:

```
command-name input-pathname-list [preposition
output-pathname-list] [other parameters]
command-entry-terminator
```

The console operator must always specifically type the "command name" entry on the keyboard to load and execute the associated program file. Except for the "command entry terminator" (RETURN, LINE FEED, or ESC key), all other command elements are optional, and the console operator may use default options when the command line is entered.

Before you create a new command, you must make a number of decisions that will affect the command line structure:

- The intended function or functions of the command.
- A command name, which you will use as the pathname for the executable program file. Ideally, the command name should be a descriptive verb or noun that has some functional meaning for the console operator who will be executing the command. COPY and TIME are examples of command names that are a descriptive verb and a descriptive noun, respectively.
- For file handling commands, an input pathname-list is usually desirable. You will use the C\$GET\$INPUT\$PATHNAME call to parse the input pathname-list.
- If output is to be performed, you will use the C\$GET\$OUTPUT\$PATHNAME call in your program to parse the preposition and output pathname-list.
- If other parameters are to be used in the new command, use the C\$GET\$PARAMETER call to parse the parameters.

CREATING NEW COMMANDS

Commands are programs that can contain system calls, including those supplied by the Human Interface. The Human Interface system calls are divided into functional groups, as follows:

- Input and output parameter calls
- Message processing calls
- Command processing calls
- Program control call
- Parameter parsing calls

A brief description of the functions performed within these groups is provided in the following subheadings; more specific information is given in the individual call descriptions in Chapter 6.

INPUT AND OUTPUT PARAMETER CALLS

Every Human Interface command (Intel-supplied or user-created) that performs I/O as part of its processing, must contain two or more of the following elements in its parameter string:

- Command name
- Input list of pathnames
- Preposition (TO/OVER/AFTER entry) and output list of pathnames.

Examples:

<u>command name</u>	<u>in-pathname-list</u>	<u>preposition</u>	<u>output destination</u>
COPY	a,b	TO	c,db
RENAME	alpha,beta	OVER	gamma,delta

The command name is usually a descriptive verb, and is a required element for every command. When any form of I/O is to be performed, a list of one or more input files must be specified. When output is to be performed, both a preposition and a list of one or more destination files must be specified to receive the output.

The Human Interface provides four system calls that you can use to process the I/O elements in a command line:

C\$GET\$INPUT\$PATHNAME

Returns a pathname or pathnames that identify the input source or sources.

CREATING NEW COMMANDS

- `CGETOUTPUT$PATHNAME` Returns a pathname or pathnames that identify the destination or destinations of the output. If the operator does not specify a preposition and pathname in a command line, `TO :CO:` (console output) is the default.
- `CGETINPUT$CONNECTION` Returns an Extended I/O System input connection to a file specified by a pathname.
- `CGETOUTPUT$CONNECTION` Returns an Extended I/O System output connection to a file specified by a pathname.

These calls are basic to I/O processing and provide a consistent interpretation for commands. They also provide you with a method of setting up the input and output connections for a command line.

Although you can use `GET$INPUT$PATHNAME` without using `GET$OUTPUT$PATHNAME`, the reverse is not true. In your source statements, a call to `GET$INPUT$PATHNAME` must always precede a call to `GET$OUTPUT$PATHNAME` for pathname parameters.

For example, assume you have written a program whose command name is `REFINE`. If the console operator enters the command line:

```
REFINE ALPHA TO BETA
```

your program calls `CGETINPUT$PATHNAME` to get the pathname `ALPHA`. `CGETINPUT$CONNECTION` is then called to get a connection to the `ALPHA` file. After getting the connection, `REFINE` performs its programmed operation and then calls `CGETOUTPUT$PATHNAME` to get the `BETA` output pathname. The program then calls `CGETOUTPUT$CONNECTION` to get a connection to the `BETA` file. Finally, `REFINE` writes the data to the output file.

Invoked programs have two "standard logical devices" available: a Standard Input and a Standard Output. You can establish and manipulate these logical devices by calling the system calls that parse the command line and establish the standard I/O connections.

The Standard Input is parsed by the `CGETINPUT$PATHNAME` system call from the input pathnames that appear after the command name on the command line. The pathnames returned by `CGETINPUT$PATHNAME` may be given to the `CGETINPUT$CONNECTION` routine to create an I/O connection. If the `CGETINPUT$CONNECTION` call finds the input file does not exist, the following message is issued to the user's console:

```
pathname, does not exist
```

The Standard Output is parsed by using the `CGETOUTPUT$PATHNAME` call in your program. The call to this routine should appear immediately after the first call to `CGETINPUT$PATHNAME`. User output redirection is achieved by using the `TO`, `OVER`, or `AFTER` prepositions, which must appear

CREATING NEW COMMANDS

immediately after the Standard Input specification on a command line. These prepositions are interpreted by the C\$GET\$OUTPUT\$PATHNAME and C\$GET\$OUTPUT\$CONNECTION routines called in your program. When accepted by a command, these prepositions are reserved words that appear in a command line with the following syntax:

```
TO
OVER      outpath-list
AFTER
```

If a TO/OVER/AFTER preposition is not found on the command line, the default supplied by the C\$GET\$OUPUT\$PATHNAME call will be used.

The pathname parameter may be either for a single file or a list of files. The TO/OVER/AFTER prepositions give the user a choice in the way a specified pathname is to be handled.

The TO preposition specified in a command line indicates that the pathname should not already exist as a maintained file before the operation is performed. If the parsing call finds the file does exist, the following message is displayed on the console screen:

```
pathname, already exists, DELETE?
```

Only a keyboard response of "Y" or "y" will cause the operation to be performed for that particular pathname in a pathname list.

The OVER preposition in a command line specifies that, if the pathname already exists, the pathname is to be deleted and command execution is to continue.

The AFTER preposition in a command line specifies that the standard output is to be appended to end of the contents of the pathname.

The Command line Interpreter (see the description in Chapter 1) provides protection for a user's files for every preposition except OVER. For example:

```
TO xz
OVER xz
AFTER xz
```

all execute without a warning message if xz does not exist. However, only the OVER preposition can replace an existing file without the warning message being issued (although AFTER can concatenate to an existing file).

MESSAGE PROCESSING CALLS

The message processing calls provide a means to send and receive messages from the user's console, and to format a default message for a specified exception code.

CREATING NEW COMMANDS

COMMAND PROCESSING CALLS

The command processing calls are intended for users who wish to create their own stream of commands or employ modifications to some of the functions performed by other Human Interface calls. There are three such calls and they are used in combination, as follows:

1. C\$CREATE\$COMMAND\$CONNECTION -- establishes a token for a command connection that allows multiple line commands without interference from other tasks. The command connection TOKEN established by the call is used in the C\$SEND\$COMMAND and C\$DELETE\$COMMAND\$CONNECTION calls.
2. C\$SEND\$COMMAND -- accepts command lines and combines them with lines previously sent. After a full command is received, C\$SEND\$COMMAND loads the command for execution.
3. C\$DELETE\$COMMAND\$CONNECTION -- deletes a command connection TOKEN previously established by a C\$CREATE\$COMMAND\$CONNECTION call and frees the memory used for the connection's data structures.

PROGRAM CONTROL CALL

The C\$SET\$CONTROL\$C call lets you modify the "standard" response of the CTRL/c function by changing the calling program's CTRL/c semaphore to some other semaphore specified in a SET\$CONTROL\$C call. An application example would be to change the CTRL/c semaphore of an edit program so that when CTRL/c was pressed, it would not cause the editor to exit but would instead allow it to cancel the current operation and prompt for a new command.

PARAMETER PARSING CALLS

The Human Interface parsing calls help ensure system consistency in processing commands. The parsing calls minimize parsing inconsistencies in user-created programs, thus making it easier for the programmer to provide command line parsing in the new program. The parameter parsing calls support two different types of parameters:

- The first type of parameter are parameters with values, which are keyword parameters and preposition parameters. Parameters with values may appear in three different forms:

keyword=value-part	(keyword parameter)
keyword(value-part)	(keyword parameter)
keyword value-part	(preposition parameter)

Note that the first two forms are called keyword parameters and the last form is called a preposition parameter, and that all three have a keyword and a value-part. The differences between the forms are the separators

CREATING NEW COMMANDS

used in each. The reason for distinguishing between keyword parameters and a preposition parameter lies in the method used to recognize them during command line parsing:

- The presence of the equal sign (=) or the pair of parentheses () lets the Command Line Interpreter recognize keyword parameters without foreknowledge of the keywords.
- The structure of the preposition parameter is so loose that the Command Line Interpreter must be provided with a list of the prepositions before they can be recognized. You must supply the CLI with this list in the form of a predict\$stable whenever a parameter is requested from the parsing routines. See Table 6-1 for a listing of the Human Interface parameter parsing calls.

The value-part of a parameter is either a single value or a list of values separated by commas. In addition, the value-part may itself contain value-lists. If enclosed in parentheses, a value-list will be returned to the STRING\$TABLE as one value, including the parentheses. For example:

A,(B,C,D),E

where (B,C,D) will be returned as one value. The string\$stable format is illustrated in Appendix C this manual.

- The second type of supported parameter are lists of values, which may consist of a single value or a list of values that comprise the value-part of a parameter. The values must be separated by commas. Note that since a value-list is considered one value, any quote marks appearing within the parentheses will not get deleted. See the C\$GET&PARAMETER system call in Chapter 6 for more information.

Parameter parsing may be performed by using high-level calls to get parameters or characters. If, instead, a command line is accessed at the character level, it is the program's responsibility to provide syntax compatibility. Application programs will not receive any comments found in the command lines.

All or a portion of a parameter can be quoted by using either single or double quote marks. The same type of quote mark must be used to end a quoted string as was used to begin the string; that is, if a quoted string began with a single quote(') it must be terminated with a single quote.

PARAMETER SYNTAX CONSIDERATIONS

Preposition parameters, such as TO/OVER/AFTER, may not be abbreviated. However, keyword parameter names may be abbreviated in one of two ways:

1. Standard Abbreviation - this is performed by typing only as many characters of the parameter as are necessary to make it unique.

CREATING NEW COMMANDS

2. Syllabic Abbreviation - this is performed by conforming to the following rules:
 - a. If a name consists of two or more concatenated words, use the first letter from each word. Digit sequences may not be abbreviated.
 - b. If a name consists of only one syllable, use the first two letters of the parameter name.
 - c. If a name consists of two or more syllables, use the first letter of the first two syllables.
 - d. If negation is used (i.e., NOLIST), the "NO" is not abbreviated; for example, NOLI.
 - e. If rules a through c result in the same abbreviation for different name elements, a third letter must be added to one of the names to achieve uniqueness in the abbreviation. However, the third letter need not be added if the conflicting names appear in different programs, or in the same program but in different contexts.

COMMAND INVOCATION

All executable programs are non-resident and are invoked as separate iRMX 86 programs. After the command name has been broken out of a command line, the CLI parses for the command pathname. If the pathname contains an explicit logical name, the pathname is used "as is". However, if the pathname consists only of components, such as a single filename, the CLI first searches the User Program directory (:PROG:), where user-created commands are cataloged. If the file is not found, the CLI then searches the System Command directory (:SYSTEM:), where the Human Interface commands are cataloged. Note that this order of search and the presence of the directories are configuration-time options. See Table 5-1 and Chapter 1 for descriptions of the :PROG: and :SYSTEM: directories.

PROGRAM CONTROL

When a user invokes a Human Interface command, the Human Interface invokes the Application Loader to load the command from secondary storage and create it as an I/O job. Whenever the calling task's priority is the highest, the command starts executing. While the command is executing, terminal is ignored and the user cannot enter other commands until the first command finishes processing. In order to finish processing, the command must contain, as its last executable statement, a call to EXIT\$IO\$JOB. Otherwise the terminal will be hung up forever.

The format of the call to EXIT\$IO\$JOB is as follows:

```
CALL RQ$EXIT$IO$JOB(0,0,excep$ptr)
```

CREATING NEW COMMANDS

This implies a normal exit with no message sent to the message mailbox and with the exception code returned in the location pointed to by `except$ptr`.

If an executing program contains a `CSETCONTROL$C` system call, an operator can cancel program execution by pressing the CTRL/c key. See the `CSETCONTROL$C` system call description in Chapter 6.

EXCEPTION HANDLER

Exceptions are divided into two categories: programming errors and environmental conditions. Programming errors occur when the iRMX 86 system detects a condition that normally can be avoided by correct coding. Environmental conditions, in contrast, are generally outside the control of the application program.

The Human Interface provides a default exception handler for each program it creates. This exception handler receives control on the occurrence of all exceptions. You can cause all exceptions or no exceptions in an application to be passed to the exception handler by using the Nucleus `CGETEXCEPTION$HANDLER` and `C$SET$EXCEPTION$HANDLER` system calls. The action taken by Human Interface exception handler will be to print an error message and then cancel the program.

Since you may not want to construct a special exception message in a user application, you can use the Human Interface `C$FORMAT$EXCEPTION` system call to construct a standard message. You can pass either programming or environmental exceptions to `C$FORMAT$EXCEPTION`, which will construct a standard message. The application program would issue the message and then either continue execution or exit itself.

LOGICAL NAMES

You may use either the logical names provided by the Human Interface or create your own at system configuration time. See the iRMX 86 CONFIGURATION GUIDE for more information. The standard logical names and their meaning are described in Table 2-1.

You can also assign logical names to new physical devices that are added to the system without going through a system reconfiguration. See the `ATTACHDEVICE` command in Chapter 3 for more information.

LOAD MODULE FORMATS

You can create new application programs in either relocatable or absolute load format. However, if you use an absolute format, you must reserve memory for the absolute object modules at iRMX 86 system configuration time. Refer either to the iAPX 86,88 FAMILY UTILITIES USER'S GUIDE

CREATING NEW COMMANDS

for 8086-BASED DEVELOPMENT SYSTEMS or to the 8086 FAMILY UTILITIES GUIDE for 8080/8085-BASED DEVELOPMENT SYSTEMS, as appropriate for your development system, and the iRMX 86 CONFIGURATION GUIDE for more information.

COMMAND CREATION PROCEDURES

The general procedures for creating a user-designed program that can be executed by keyboard command are as follows:

1. Assuming you will be creating the program on a development system, write the program code.
2. Assemble or compile the program.
3. Link the program to the appropriate iRMX 86 libraries, including the Human Interface library. The link parameters to use depend on your type of development system and the type of load module desired, as follows:

Series III, Alternative A - creates a relocatable object module in LTL or PIC code. Use LINK86 with the SEGSIZE (STACK(u)), MEMPOOL, and BIND controls to generate a relocatable object module. Generating an relocatable object module is recommended to avoid having to reserve memory at system configuration time.

Series III, Alternative B - creates an absolute object module. Use the LINK86 and LOC86 commands and the NOINITCODE and SEGSIZE (STACK(u)) controls to generate the absolute code. You must reserve memory space at system configuration time to receive the absolute object module.

Series II - creates an absolute object module (only). Use the LINK86 and LOC86 commands with SEGSIZE (STACK(u)) control. You must reserve memory space at system configuration time to receive the absolute module.

4. Copy the program module from the development system to the :PROG: directory on the appropriate iRMX 86 diskette by using the Human Interface UPCOPY command.
5. Load and execute the program by typing its command name on the user console.

See either the iAPX 86,88 FAMILY UTILITIES USER'S GUIDE for 8086-BASED DEVELOPMENT SYSTEMS or the 8086 FAMILY UTILITIES USER'S GUIDE for 8080/8085-BASED DEVELOPMENT SYSTEMS, as appropriate for your development system, for more information about the LINK86 and LOC86 commands.

CHAPTER 6. HUMAN INTERFACE SYSTEM CALLS

The Human Interface system calls described in this chapter are presented in alphabetical sequence without regard to functional organization. A functional grouping of the calls according to type is provided in the System Call Dictionary in Table 6-1. For each call, the information is organized into the following categories:

- Brief functional description.
- Calling sequence format.
- Input parameter definitions, if applicable.
- Output parameter definitions, if applicable.
- Considerations and consequences of call usage.
- Potential exception codes, and their possible causes.

Throughout the call descriptions, iRMX 86 data types such as WORD and STRING are used. The data types are always capitalized, and they are defined in Appendix C.

If you are a new user of the Human Interface calls, it is suggested that you review the calling considerations in Chapter 5 before implementing these command processing routines in your source code. Quick review of the command format and syntax descriptions in Chapter 2, and file handling examples in Chapter 4 may also be useful.

It is also assumed that you are familiar with a number of terms and concepts that are common to the iRMX 86 Operating System. If you are not, it is suggested that you take the time to read INTRODUCTION TO THE iRMX 86 OPERATING SYSTEM and the chapters in the iRMX 86 NUCLEUS REFERENCE MANUAL that refer to the terms memory pool and catalog.

HUMAN INTERFACE SYSTEM CALLS

Table 6-1. System Call Dictionary

System Call	Synopsis	Page
I/O Processing Calls		
C\$GET\$INPUT\$CONNECTION	Return an EIOS connection for the specified input file.	6-14
C\$GET\$OUTPUT\$CONNECTION	Return an EIOS connection for the specified output file.	6-22
Command Parsing Calls		
C\$GET\$CHAR	Get a character from the command line	6-13
C\$GET\$INPUT\$PATHNAME	Parse the command line return a pathname that will identify the Standard Input file.	6-20
C\$GET\$PARAMETER	Parse the command line for the next parameter and return it as a keyword name and a value.	6-34
C\$GET\$OUTPUT\$PATHNAME	Parse the command line and return a pathname that will identify the Standard Output file.	6-31
C\$SET\$PARSE\$BUFFER	Parse a buffer other than the current command line.	6-56
Message Processing Calls		
C\$FORMAT\$EXCEPTION	Format a default message into a user buffer for a given exception code.	6-11
C\$SEND\$CO\$RESPONSE	Send a message to the command output (CO) and read a response from the command input (CI).	6-48
C\$SEND\$EO\$RESPONSE	Send a message to the error output device (EO) and return a response from the error input device (EI).	6-51

HUMAN INTERFACE SYSTEM CALLS

Table 6-1. System Call Dictionary (continued)

System Call	Synopsis	Page
Command Processing Calls		
C\$CREATE\$COMMAND\$CONNECTION	Create a command connection and return a token.	6-4
C\$DELETE\$COMMAND\$CONNECTION	Delete a specific command connection.	6-10
C\$SEND\$COMMAND	Concatenate command lines into the data structure created by CREATE\$COMMAND\$CONNECTION and then execute command.	6-38
Program Control Call		
C\$SET\$CONTROL\$C	Change calling program's CONTROL C semaphore to the specified semaphore.	6-54

HUMAN INTERFACE SYSTEM CALLS

C\$CREATE\$COMMAND\$CONNECTION

C\$CREATE\$COMMAND\$CONNECTION, a command processing call, establishes a **TOKEN** for a command connection that can be used to combine and invoke multiple-line commands without interference from other tasks.

```
command$conn = RQ$C$CREATE$COMMAND$CONNECTION(default$ci, default$co,
                                              reserved$word,
                                              except$ptr);
```

INPUT PARAMETERS

default\$ci	A WORD containing a TOKEN representing a connection that will be used as the :CI: (console input) for any command started using this command connection.
default\$co	A WORD containing a TOKEN representing a connection that will be used as the :CO: for any commands started using this command connection.
reserved\$word	A WORD reserved for future use. Its value should be zero (0).

OUTPUT PARAMETERS

command\$conn	A WORD which receives a TOKEN for a command connection that subsequently may be used by C\$SEND\$COMMAND and C\$DELETE\$COMMAND\$CONNECTION to refer to a particular command stream.
except\$ptr	A POINTER to a WORD in which the Human Interface will return an exception code.

DESCRIPTION

The call creates a data structure and returns a **TOKEN** for a command connection that may be used in conjunction with the **C\$SEND\$COMMAND** call. It also contains the default **:CI:** and **:CO:** connection that must be defined by the caller for commands invoked by **C\$SEND\$COMMAND** calls.

This call can be used to invoke a command programmatically instead of interactively; that is, one command can invoke still another command for execution, and so on.

EXCEPTION CODES

- E\$OK** No exceptional conditions were encountered.
- E\$CONTEXT** When your task invoked this system call, the call invoked at least one other system call. While the Operating System performed the latter call, one of the following situations occurred.
- The Operating System detected a zero value for the object directory size. This condition occurred because an improper configuration of the Human Interface subsystem set the job object directory size to zero.
 - The Operating System detected two command connections being created simultaneously by two tasks in the same job. This condition occurred because a programmer miscalculated or disrupted a synchronized use of the command connection.
 - The Operating System detected the :STREAM: device in the process of being detached.
 - The job containing the task which invoked this system call was not created by the Human Interface subsystem.
 - While creating a STREAM file, the Extended I/O System was unable to attach the :STREAM: device because another task had already invoked a Basic I/O system call to attach the :STREAM: device.
 - The Basic I/O System was unable to attach the default\$ci or default\$co connection parameters. This occurred because the Basic I/O System was detaching the device on which the connection was based.
- E\$DEVFD** Your task forced the Extended I/O System to attempt the physical attachment of :STREAM: that had formerly been only logically attached. In the process of attempting to physically attach :STREAM:, the Extended I/O System found that the device and the device driver specified in the logical attachment were incompatible. The Operating System would not have returned this exception code if :STREAM: had been properly configured in the Extended I/O and/or Basic I/O subsystems.

C\$CREATE\$COMMAND\$CONNECTION

EXCEPTION CODES (continued)

- E\$EXIST** When your task invoked this system call, the call invoked at least one other system call. While the Operating System performed the latter call, one of the following situations occurred.
- This condition occurred when your task used a Human Interface call but was not within a job created by the Human Interface subsystem.
 - The parameter default\$ci or default\$co was not a token for a valid object.
 - The Basic I/O System deleted either the parameter default\$ci or default\$co.
- E\$I/O** While attempting to attach the parameter default\$ci or default\$co, the Basic I/O System detected an I/O error.
- E\$IOMEM** The Basic I/O System job does not currently have a block of memory large enough to allow the Human Interface to create a stream file.
- E\$LIMIT** When your task invoked this system call, the call invoked at least one other system call. While the Operating System performed the latter call, one of the following situations occurred.
- While creating the objects needed by this call, the Operating System detected the calling task's job as already having reached the maximum number of objects that can exist simultaneously. This condition occurred because an improper configuration of the Human Interface subsystem set the number of the maximum objects within the job too low.
 - The Operating System detected the object directory of the calling task's job as already reached the maximum object directory size.
 - The job containing the task which invoked this call or the job's default user object is currently involved in more than 255 (decimal) I/O operations.

EXCEPTION CODES

E\$LIMIT (continued)

- The job containing the task which invoked this call was not created by the Human Interface subsystem.
- The object limit of the Basic I/O System job has been exceeded. Refer to the chapter of the iRMX 86 CONFIGURATION GUIDE that discusses the Basic I/O System.
- During the process of configuring your application system, the Basic I/O System job was configured a maximum priority that is too low. Specifically, the BIOS maximum priority is lower than either the DUIB priority or the DEVINFO priority. Refer to the iRMX 86 CONFIGURATION GUIDE for information regarding the BIOS maximum priority. Refer to the GUIDE TO WRITING DEVICE DRIVERS FOR THE iRMX 86 AND iRMX 88 I/O SYSTEMS for additional information regarding the DUIB and DEVINFO.

E\$LOG\$NAME
\$NEXIST

The :STREAM: device does not exist. This condition occurred because someone failed to configure :STREAM: during the configuration of the Extended I/O or Basic I/O subsystems.

E\$MEM

The memory pool of the job whose task invoked this call does not currently have a block of memory large enough to allow this system call to run to completion.

E\$NOT\$CONFIGURED

One or more of the following system calls are not configured into the system:

<u>Call</u>	<u>Subsystem</u>
A\$ATTACH\$FILE	Basic I/O
A\$CREATE\$FILE	Basic I/O
A\$GET\$FILE\$STATUS	Basic I/O
A\$OPEN	Basic I/O
A\$PHYSICAL\$ATTACH\$DEVICE	Basic I/O
A\$SPECIAL	Basic I/O
CATALOG\$OBJECT	Nucleus
CREATE\$COMPOSITE	Nucleus
CREATE\$MAILBOX	Nucleus
CREATE\$SEGMENT	Nucleus
CREATE\$TASK	Nucleus
CREATE\$SEMAPHORE	Nucleus
DELETE\$COMPOSITE	Nucleus

C\$CREATE\$COMMAND\$CONNECTION

EXCEPTION CODES

E\$NOT\$CONFIGURED (continued)

	<u>Call</u>	<u>Subsystem</u>
	DISABLE\$DELETION	Nucleus
	ENABLE\$DELETION	Nucleus
	GET\$DEFAULT\$PREFIX	Basic I/O
	GET\$TYPE	Nucleus
	LOOKUP\$OBJECT	Nucleus
	RECEIVE\$CONTROL	Nucleus
	RECEIVE\$MESSAGE	Nucleus
	S\$CREATE\$FILE	Extended I/O
	SEND\$CONTROL	Nucleus
	SEND\$MESSAGE	Nucleus
	SET\$INTERRUPT	Nucleus
	S\$GET\$CONNECTION\$STATUS	Extended I/O
	WAIT\$INTERRUPT	Nucleus
E\$NOT\$PREFIX	The :STREAM: logical name does not refer to either a device connection or a file connection.	
E\$NO\$USER	When your task invoked this system call, the call invoked at least one other system call. While the Operating System performed the latter call, one of the following situations occurred. <ul style="list-style-type: none">• The job containing the task which invoked this call does not have a default user. This indicates that your task used a Human Interface call but was not within a job created by the Human Interface subsystem.• The job containing the task which invoked this call has a default user. This default user is not a user object.• The system call LOOKUP\$OBJECT was not configured.	
E\$PARAM	The Extended I/O System was forced to attempt the physical attachment of a :STREAM: device that had formerly been only logically attached. In the process of attempting to physically attach the :STREAM: device, the Extended I/O System found that the logical attachment referred to a file driver (named, physical, or stream) that is not configured into your system. Hence the physical attachment is not possible. The Operating System would not have returned this exception code if the :STREAM: device had been properly configured in the Extended I/O and/or Basic I/O subsystems.	

EXCEPTION CODES (continued)

E\$SUPPORT	When your task invoked this system call, the call invoked at least one other system call. While the Operating System performed the latter call, one of the following situations occurred.
	<ul style="list-style-type: none">● During the process of configuring the Basic I/O subsystem, someone improperly configured the Named File Driver Table. This table is divided into a request part and an I/O system part. A\$ATTACH\$FILE or A\$OPEN was configured in the request part, but the corresponding entry in the I/O system part was not included. Refer to the iRMX 86 CONFIGURATION GUIDE for further information.● The %NO_CREATE_FILE or the %NO_TRUNCATE macro calls were configured into the Basic I/O subsystem.
E\$TIME	Your task used a Human Interface call but was not within a job created by the Human Interface subsystem.
E\$TYPE	When your task invoked this system call, the call invoked a file-attaching call. While performing the file-attaching call, the Operating System detected a default\$ci or default\$co parameter that is not a connection.

C\$DELETE\$COMMAND\$CONNECTION

C\$DELETE\$COMMAND\$CONNECTION deletes a command connection object and frees the memory used by the connection's data structures.

```
CALL RQSC$DELETE$COMMAND$CONNECTION(command$conn, except$ptr);
```

INPUT PARAMETER

command\$conn A WORD containing a token for a valid command connection.

OUTPUT PARAMETER

except\$ptr A POINTER to a WORD in which the Human Interface will return an exception code.

DESCRIPTION

This call deletes a command connection object previously defined in a C\$CREATE\$COMMAND\$CONNECTION call and releases the memory used by the connection's data structures.

EXCEPTION CODES

E\$OK No exceptional conditions were encountered.

E\$TYPE The command connection parameter refers to an object that is not a command connection object.

C\$FORMAT\$EXCEPTION

C\$FORMAT\$EXCEPTION, a message processing call, builds a default message for a given exception code into a user-provided buffer.

```
CALL RQSC$FORMAT$EXCEPTION(buff$p, buff$max, exception$code, brevity,
                           except$ptr);
```

INPUT PARAMETERS

buff\$p	A POINTER to a buffer that will receive a STRING from the Human Interface containing a formatted exception message.
buff\$max	A WORD that specifies the maximum number of bytes that may be contained in the buffer pointed to by buff\$p.
exception\$code	A WORD containing the exception code value for which a message is to be created.
brevity	A required BYTE that must be a "1". This gives the format of the message as follows: xxxx: (exception name) up to a total length of 30 characters. The exception name is provided by the Human Interface from an internal table.

OUTPUT PARAMETER

except\$ptr	A POINTER to a WORD in which the Human Interface will return an exception code.
-------------	---

DESCRIPTION

C\$FORMAT\$EXCEPTION causes the Human Interface to format a message for the exception code. The call concatenates the message to the end of the STRING already in the buffer. If a STRING is not already present in the buffer, the first byte of the buffer must be a zero. The message added by C\$FORMAT\$EXCEPTION will not be longer than 79 characters. The call will handle both environmental and programmer error exception codes.

C\$FORMAT\$EXCEPTION

EXCEPTION CODES

E\$OK	No exceptional conditions were encountered.
E\$PARAM	An unknown exception code was given.
E\$STRING\$BUFFER	Someone specified a buffer that is too small. This buffer is pointed to by the buff\$p parameter. This buffer provides for the exception message and is not large enough to contain the message.

C\$GET\$CHAR

C\$GET\$CHAR, a command parsing call, gets a character from the command line.

```
char = RQ$C$GET$CHAR(except$ptr);
```

OUTPUT PARAMETERS

char	A BYTE containing the next character in the command line's character string. A null (OOH) character will be returned when there are no more characters.
except\$ptr	A POINTER to a WORD in which the Human Interface will return an exception code.

DESCRIPTION

Consecutive calls to GET\$CHAR get consecutive characters from the command line. All parsing of the command line must be done in one task. When using C\$GET\$CHAR in commands, you should ensure that the commands using it are syntactically compatible with other commands provided by the Human Interface. Note that use of C\$GET\$CHAR decreases the possibility that the command syntax will remain compatible with future Human Interface releases.

EXCEPTION CODES

E\$OK	No exceptional conditions were encountered.
E\$CONTEXT	The Operating System detected a zero value for the object directory size. This indicated that your task used a Human Interface call but was not within a job created by the Human Interface subsystem.
E\$NOT\$CONFIGURED	The Nucleus system call LOOKUP\$OBJECT was not incorporated during system configuration.
E\$TIME	Your task used a Human Interface call but was not within a job created by the Human Interface subsystem.

C\$GET\$INPUT\$CONNECTION

C\$GET\$INPUT\$CONNECTION, an I/O processing call, returns an Extended I/O System connection to the specified input file.

```
connection = RQ$C$GET$INPUT$CONNECTION(name$p, except$ptr);
```

INPUT PARAMETER

name\$p A POINTER to a STRING containing the pathname of the file to be accessed.

OUTPUT PARAMETERS

connection A WORD that receives the token for the T\$CONNECTION object for the specified pathname.

except\$ptr A POINTER to a WORD in which the Human Interface will return an exception code.

DESCRIPTION

The returned input connection will be open for reading and will have the following attributes:

- Read only
- Accessible to all
- Two 1024-byte buffers

C\$GET\$INPUT\$CONNECTION may display one of the following messages on :CO: if an unusual or disallowed operation is encountered:

pathname, file not found

meaning that the input file does not exist, or:

pathname, READ access required

meaning that the user is not allowed read access to the input file.

DESCRIPTION (continued)

C\$GET\$INPUT\$CONNECTION causes an error message to be displayed whenever an exceptional condition is encountered. The exceptional condition that triggers the error message can be either one of those listed for C\$GET\$INPUT\$CONNECTION or from the Extended I/O System calls S\$ATTACH\$FILE and S\$OPEN.

The error message has the form:

pathname, xxxx:exception code

where:

pathname	Pathname of the file or directory that was being accessed when the exceptional condition was detected.
xxxx	Four-digit value of the exception code.
exception code	One of the exception codes defined under "Exception Codes" in this call description.

EXCEPTION CODES

E\$OK	No exceptional conditions were encountered.
E\$CONTEXT	<p>When your task invoked this system call, the Human Interface call invoked at least one other system call. While the Operating System performed the latter call, one of the following situations occurred.</p> <ul style="list-style-type: none"> ● The Operating System detected a zero value for the object directory size. This indicates that your task used a Human Interface call but was not within a job created by the Human Interface subsystem. ● The Basic I/O subsystem determined that the device referenced by the name\$p parameter was in the process of being detached. ● The job containing the task which invoked this system call was not created by the Human Interface subsystem. ● While processing a file-attaching system call, the Extended I/O System was unable to attach the device containing the file, referenced by the name\$p parameter. This occurred because the Basic I/O System has already attached the device.

EXCEPTION CODES

E\$DEVFD	The Extended I/O System was forced to attempt the physical attachment of a device that had formerly been only logically attached. In the process of attempting to physically attach the device, the Extended I/O System found that the device and the device driver specified in the logical attachment were incompatible. The Operating System would not have returned this exception code if the device referenced by the name\$p parameter had been properly configured in the Extended I/O and/or the Basic I/O subsystems.
E\$FACCESS	The access rights embedded in the connection have prohibited you from opening the file in the read mode. This exceptional condition can arise only when the connection refers to named files.
E\$FNEXIST	While attaching the file specified in the name\$p parameter, the Operating System detected one of the following circumstances: <ul style="list-style-type: none">• Either some file in the specified pathname (referenced by the name\$p parameter), or the target file itself, is marked for deletion.• Either some file in the specified pathname (referenced by the name\$p parameter), or the target file itself, does not exist.
E\$FTYPE	The Operating System detected an error in the pathname specified by the name\$p parameter. The pathname included the name of a data file as a directory. For example, the pathname A/B/C assumes that A and B are names for directories. This exception code would have been returned if either A or B was actually a data file.
E\$ILLVOL	While attaching the file pointed to by the name\$p parameter, the file attaching call forced the Extended I/O System to attempt the physical attachment of the device as a named device. This device had formerly been only logically attached. In the process of attempting to physically attach the device, the Extended I/O System examined the volume label and found that the volume does not contain named files. This prevented the Extended I/O System from completing physical attachment because the named file driver was requested during logical attachment.

EXCEPTION CODES (continued)

E\$IO	An I/O error occurred while trying to access the file given in the name\$p parameter.
E\$IOMEM	While attempting to create a connection, memory from the Basic I/O subsystem's memory pool was needed. However, the Basic I/O System job does not currently have a block of memory large enough to allow this system call to run to completion.
E\$LIMIT	<p>When your task invoked this system call, the call invoked at least one other system call. While the Operating System performed the latter call, one of the following situations occurred.</p> <ul style="list-style-type: none"> • The Extended I/O subsystem created enough objects to exceed the object limit of the Basic I/O System job. Refer to the chapter of the iRMX 86 CONFIGURATION GUIDE that discusses the Basic I/O System. • During the process of configuring your application system, the Basic I/O System job was configured a maximum priority that is too low. Specifically, the BIOS maximum priority is lower than either the DUIB priority or the DEVINFO priority. Refer to the iRMX 86 CONFIGURATION GUIDE for information regarding the BIOS maximum priority. Refer to the GUIDE TO WRITING DEVICE DRIVERS FOR THE iRMX 86 AND iRMX 88 I/O SYSTEMS for additional information regarding the DUIB and DEVINFO. • While processing a file-attaching system call or connection opening system call, the Operating System detected either of the two following situations: the calling task's job or the job's default user object is currently involved in more than 255 (decimal) I/O operations. • The job containing the task which invoked this system call was not an I/O job created by the Human Interface subsystem.
E\$LOG\$NAME \$NEXIST	The specified pathname for the specified device (referenced by the name\$p parameter) contains an explicit logical name. The Extended I/O System, however, was unable to find this name in the object directories of the local job, the global job, and the root job.
E\$MEDIA	The Operating System detected that the device containing the specified file (referenced by the name\$p parameter) was not online.

C\$GET\$INPUT\$CONNECTION

EXCEPTION CODES (continued)

E\$MEM The memory pool of the calling task's job as not currently having a block of memory large enough to allow this system call to run to completion.

E\$NO\$PREFIX The pathname specified in the name\$p parameter of this call contained no explicit prefix (no logical name), so the Extended I/O System attempted to use the default prefix. However, the default prefix is either undefined, or it is not a valid device connection or file connection.

E\$NOT\$CONFIGURED There are two possible conditions that can cause the Human Interface System to return this call:

- When your task invoked this system call, it forced the Extended I/O System to attempt the physical attachment of the device referenced by the name\$p parameter. This device had formerly been only logically attached. In the process of attempting to physically attach the device, the Extended I/O System found that the logical attachment referred to a file driver (named, physical, or stream) that was not configured into your system.
- At least one of the following system calls was left out of the system during the configuration process:

<u>Call</u>	<u>Subsystem</u>
A\$ATTACH\$FILE	Basic I/O
A\$GET\$FILE\$STATUS	Basic I/O
A\$OPEN	Basic I/O
A\$PHYSICAL\$ATTACH\$DEVICE	Basic I/O
A\$SPECIAL	Basic I/O
CREATE\$COMPOSITE	Nucleus
CREATE\$MAILBOX	Nucleus
CREATE\$SEGMENT	Nucleus
DELETE\$COMPOSITE	Nucleus
DISABLE\$DELETION	Nucleus
ENABLE\$DELETION	Nucleus
GET\$DEFAULT\$PREFIX	Basic I/O
GET\$TYPE	Nucleus
LOOKUP\$OBJECT	Nucleus
RECEIVE\$CONTROL	Nucleus
RECEIVE\$MESSAGE	Nucleus
S\$ATTACH\$FILE	Extended I/O
SEND\$CONTROL	Nucleus
SEND\$MESSAGE	Nucleus
SET\$INTERRUPT	Nucleus
S\$OPEN	Extended I/O
WAIT\$INTERRUPT	Nucleus

EXCEPTION CODES (continued)

E\$NOT\$PREFIX	The pathname specified by the name\$p parameter contained a logical name. The name referred to an object that was neither a device connection nor a file connection.
E\$NO\$USER	The job containing the task which invoked this call does not have a default user or the default user of this calling task's job was not a user object.
E\$PARAM	<p>When your task invoked this system call, the call invoked at least one other system call. While the Operating System performed the latter call, one of the following situations occurred.</p> <ul style="list-style-type: none"> • The pathname specified by the name\$p parameter contained a logical name. The name was either longer than 12 characters or contained invalid characters. • The system call forced the Extended I/O System to attempt the physical attachment of the device referenced by the name\$p parameter. The device had formerly been only logically attached. In the process of attempting to physically attach the device, the Extended I/O System found that the logical attachment referred to a file driver (named, physical, or stream) that is not configured into your system. Hence the physical attachment is not possible.
E\$PREFIX\$SYNTAX	The Operating System detected a colon (:) at the start of the pathname specified by the name\$p parameter. This indicates that the pathname contains a logical name. However, the Operating System was unable to find a second colon to terminate the logical name.
E\$SHARE	The Operating System detected some other task in your system using the I/O System to manipulate the file through another connection. That task requested that the I/O System restrict the sharing of the file to certain modes. Your task attempted to use a read only mode that precludes sharing the file.
E\$TIME	Your task used a Human Interface call but was not within a job created by the Human Interface subsystem.

C\$GET\$INPUT\$PATHNAME

C\$GET\$INPUT\$PATHNAME, an I/O processing call, returns a pathname that may be used as the Standard Input.

```
CALL RQ$C$GET$INPUT$PATHNAME(path$name$p, path$name$max, except$ptr);
```

INPUT PARAMETER

path\$name\$max A WORD that specifies the length in bytes of the string pointed to by the **path\$name\$p** parameter, where the maximum length that may be specified for a pathname is 255 characters, plus one extra byte.

OUTPUT PARAMETERS

path\$name\$p A POINTER to a buffer that will receive a STRING containing the next pathname in the Standard Input. A zero-length string indicates that there are no more pathnames.

except\$ptr A POINTER to a WORD in which the Human Interface will return an exception code.

DESCRIPTION

The pathname returned by **C\$GET\$INPUT\$PATHNAME** may be used for the following purposes:

- In calls to the Extended I/O System.
- In a call to **C\$GET\$INPUT\$CONNECTION**, to obtain a connection.

The pathname will be a single pathname from the command line's input list. The next file in the input sequence may then be requested by making successive calls to **C\$GET\$INPUT\$PATHNAME**. The end of the pathname-list is denoted by a zero-length string in the **path\$name\$p** buffer.

EXCEPTION CODES

E\$OK No exceptional conditions were encountered.

EXCEPTION CODES (continued)

E\$CONTEXT	The Operating System detected a zero value for the object directory size. This indicates that your task used a Human Interface call but was not within a job created by the Human Interface subsystem.
E\$EXTRA\$SO	There were no more input pathnames although the output pathname list was not empty. For example, file E is considered an extra pathname in the command entry A,B TO C,D,E.
E\$LIMIT	While creating an object, the Operating System detected a job's object limit having been exceeded. The job contained the task which invoked this system call.
E\$LIST	The last value of the input pathname list is missing. For example, ABLE,BAKER, has no value following the second comma.
E\$MEM	The memory pool of the calling task's job as not currently having a block of memory large enough to allow this latter system call to run to completion.
E\$NOT\$CONFIGURED	One or both of the following Nucleus System calls were not incorporated during system configuration: LOOKUP\$OBJECT CREATE\$SEGMENT
E\$PARSE\$TABLES	The Human Interface subsystem detected an error that should not occur unless someone inadvertently alters an internal table used by this subsystem.
E\$STRING	The pathname to be returned exceeds the length limit of 255 characters.
E\$STRING\$BUFFER	Someone specified a buffer that is too small. This buffer was pointed to by the path\$name\$p parameter and was not large enough for the pathname to be returned.
E\$TIME	Your task used a Human Interface call but was not within a job created by the Human Interface subsystem.

C\$GET\$OUTPUT\$CONNECTION

C\$GET\$OUTPUT\$CONNECTION, an I/O processing call, parses the command line and returns an Extended I/O System connection referring to the requested output file.

```
connection = RQ$C$GET$OUTPUT$CONNECTION(name$p, preposition,
                                          except$ptr);
```

INPUT PARAMETERS

name\$p A POINTER to a STRING containing the pathname of the file to be accessed.

preposition A BYTE that defines which preposition to use to create the output file. Use one of the following values to specify the preposition mode:

<u>Value</u>	<u>Meaning</u>
0	Use same preposition as was returned by the last GET\$OUTPUT\$PATHNAME call.
1	TO
2	OVER
3	AFTER
4-255	error

OUTPUT PARAMETERS

connection A WORD containing a token for an Extended I/O System connection object for the file specified when the call is invoked.

except\$ptr A POINTER to a word in which the Human Interface will return an exception code.

DESCRIPTION

The returned output connection will be open for writing and will have the following attributes:

- Write only
- Accessible to all
- Two 1024-byte buffers

C\$GET\$OUTPUT\$CONNECTION may issue one of the following prompts on :CO: if the user attempts to perform an unexpected or disallowed operation:

pathname, already exists, DELETE?

meaning that the TO preposition was used and the output file already exists, or:

pathname, DELETE access required

meaning that the pre-existing file or directory cannot be deleted without delete access to the file or directory, or:

pathname, directory ADD entry access required

meaning that the file is a directory and the user does not have add entry access to it.

C\$GET\$OUTPUT\$CONNECTION causes an error message to be displayed whenever an exceptional condition is encountered. The exceptional condition that triggers the error message can be either one of those listed for C\$GET\$OUTPUT\$CONNECTION or from an Extended I/O System call.

The error message has the form:

pathname, xxxx:exception code name

where:

pathname	Pathname of the file or directory that was being accessed when the exceptional condition was detected.
xxxx	Four-digit value of the exception code.
exception code name	One of the exception codes defined under "Exception Codes" in this call description.

C\$GET\$OUTPUT\$CONNECTION

EXCEPTION CODES

- E\$OK No exceptional conditions were encountered.
- E\$CONTEXT When your task invoked this system call, the call invoked at least one other system call. While the Operating System performed the latter call, one of the following situations occurred.
- The Operating System detected a zero value for the object directory size. This indicates that your task used a Human Interface call but was not within a job created by the Human Interface subsystem.
 - The device referenced by the name\$p parameter was in the process of being detached.
 - The Extended I/O System was unable to attach the device containing the file because the Basic I/O System has already attached the device.
- E\$DEVFD A system call forced the Extended I/O System to attempt the physical attachment of the device referenced by the name\$p parameter. This device had formerly been only logically attached. In the process of attempting to physically attach the device, the Extended I/O System found that the device and the device driver specified in the logical attachment were incompatible. The Operating System would not have returned this exception code if the device referenced by the name\$p parameter had been properly configured in the Extended I/O and/or the Basic I/O subsystems.
- E\$FACCESS When your task invoked this system call, the call invoked at least one other system call. While the Operating System performed the latter call, one of the following situations occurred.
- While in the process of creating a file, the Operating System detected the user as not having update access to an existing file and/or as not having add-entry access to the parent directory.
 - While performing a connection opening call, the access rights embedded in the connection have prohibited you from opening the file in write mode. This exceptional condition can arise only when the connection refers to a named data file or directory and the file is already open.

EXCEPTION CODES

E\$FACCESS (continued)

- While performing a system call that removes information from the end of a named data file, the Operating System detected that the user does not have update access to the file.

E\$FNEXIST

The Operating System detected one of the following circumstances:

- Either some file in the pathname specified by the name\$p parameter is marked for deletion.
- Either some file in the pathname specified by the name\$p parameter, or the target file itself, does not exist.

E\$FTYPE

The Operating System detected an error in the pathname specified by the name\$p parameter. The pathname included the name of a data file as a directory. For example, the pathname A/B/C assumes that A and B are names for directories. This exception code would have been returned if either A or B was actually a data file.

E\$ILLVOL

When the Operating System performed the call that attaches or creates a file, the call forced the Extended I/O System to attempt the physical attachment of the device referenced by the name\$p parameter. This device had formerly been only logically attached. In the process of attempting to physically attach the device, the Extended I/O System examined the volume label and found that the volume does not contain named files. This prevented the Extended I/O System from completing physical attachment because the named file driver was requested during logical attachment.

E\$IO

An I/O error occurred while trying to access or create the file given in the name\$p parameter.

E\$IOMEM

The memory required to create a connection was part of a Basic I/O System memory pool. However, the Basic I/O System job does not currently have a block of memory large enough to allow this system call to run to completion.

E\$LIMIT

When your task invoked this system call, the call invoked at least one other system call. While the Operating System performed the latter call, one of the following situations occurred.

EXCEPTION CODES

E\$LIMIT (continued)

- While creating the objects needed by this call, the Operating System detected the calling task's job as already having reached the maximum number of objects that can exist simultaneously.
- While processing the latter system call, the Operating System detected either of the two following situations: the calling task's job, or the job's default user object, is currently involved in more than 255 (decimal) I/O operations.
- The job containing the task which invoked this call was not created by the Human Interface subsystem.
- A subsystem created enough objects to exceed the object limit of the Basic I/O System job. Refer to the chapter of the iRMX 86 CONFIGURATION GUIDE that discusses the Basic I/O System.
- During the process of configuring your application system, the Basic I/O System job was configured a maximum priority that is too low. Specifically, the BIOS maximum priority is lower than either the DUIB priority or the DEVINFO priority. Refer to the iRMX 86 CONFIGURATION GUIDE for information regarding the BIOS maximum priority. Refer to the GUIDE TO WRITING DEVICE DRIVERS FOR THE iRMX 86 AND iRMX 88 I/O SYSTEMS for additional information regarding the DUIB and DEVINFO.

E\$LOG\$NAME
\$NEXIST

While processing a file-attaching call, the Operating System detected the following: the pathname specified in the name\$p parameter contains an explicit logical name. The Extended I/O System, however, was unable to find this name in the object directories of the local job, the global job, and the root job.

E\$MEDIA

The device specified by the name\$p parameter was not online.

EXCEPTION CODES (continued)

E\$MEM When your task invoked this call, the call invoked at least one other system call. While processing this latter call, the Operating System detected the memory pool of the calling task's job as not currently having a block of memory large enough to allow this system call to run to completion. To be more specific, if the latter call was attempting to process a file-attaching system call, a file-creating system call, or a connection opening system call, then the creation or the opening of a connection was not completed.

E\$NO\$PREFIX The pathname specified by the name\$p parameter of this call contained no explicit prefix (no logical name), so the Extended I/O System attempted to use the default prefix. However, the default prefix is either undefined, or it is not a valid device connection or file connection.

E\$NOT\$CONFIGURED There are two possible conditions that can cause the Human Interface System to return this call:

- When your task invoked this system call, it forced the Extended I/O System to attempt the physical attachment of the device referenced by the name\$p parameter. This device had formerly been only logically attached. In the process of attempting to physically attach the device, the Extended I/O System found that the logical attachment referred to a file driver (named, physical, or stream) that was not configured into your system.
- At least one of the following system calls was left out of the system during the configuration process:

<u>Call</u>	<u>Subsystem</u>
A\$ATTACH\$FILE	Basic I/O
A\$CREATE\$FILE	Basic I/O
A\$GET\$FILE\$STATUS	Basic I/O
A\$OPEN	Basic I/O
A\$PHYSICAL\$ATTACH\$DEVICE	Basic I/O
A\$READ	Basic I/O
A\$SEEK	Basic I/O
A\$SPECIAL	Basic I/O
A\$TRUNCATE	Basic I/O
A\$WRITE	Basic I/O
CREATE\$COMPOSITE	Nucleus
CREATE\$MAILBOX	Nucleus

C\$GET\$OUTPUT\$CONNECTION

EXCEPTION CODES

E\$NOT\$CONFIGURED (continued)

	<u>Call</u>	<u>Subsystem</u>
	CREATE\$SEGMENT	Nucleus
	DELETE\$COMPOSITE	Nucleus
	DISABLE\$DELETION	Nucleus
	ENABLE\$DELETION	Nucleus
	GET\$DEFAULT\$PREFIX	Basic I/O
	GET\$TYPE	Nucleus
	LOOKUP\$OBJECT	Nucleus
	RECEIVE\$CONTROL	Nucleus
	RECEIVE\$MESSAGE	Nucleus
	S\$ATTACH\$FILE	Extended I/O
	S\$CREATE\$FILE	Extended I/O
	SEND\$CONTROL	Nucleus
	SEND\$MESSAGE	Nucleus
	SET\$INTERRUPT	Nucleus
	S\$GET\$CONNECTION\$STATUS	Extended I/O
	S\$OPEN	Extended I/O
	S\$READ\$MOVE	Extended I/O
	S\$SEEK	Extended I/O
	S\$TRUNCATE\$FILE	Extended I/O
	S\$WRITE\$MOVE	Extended I/O
	WAIT\$INTERRUPT	Nucleus
E\$NOT\$PREFIX	The pathname specified by the name\$p parameter contained a logical name. The name referred to an object that was neither a device connection nor a file connection.	
E\$NO\$USER	Your task used a Human Interface call but was not within a job created by the Human Interface subsystem.	
E\$PARAM	When your task invoked this system call, the call invoked at least one other system call. While the Operating System performed the latter call, one of the following situations occurred. <ul style="list-style-type: none">● The Operating System detected the pathname specified by the name\$p parameter containing a logical name. The logical name was either longer than 12 characters or contained invalid characters.	

EXCEPTION CODES

E\$PARAM (continued)

- The system call forced the Extended I/O System to attempt the physical attachment of the device referenced by the name\$p parameter. The device had formerly been only logically attached. In the process of attempting to physically attach the device, the Extended I/O System found that the logical attachment referred to a file driver (named, physical, or stream) that is not configured into your system. Hence the physical attachment is not possible.

E\$PREFIX\$SYNTAX

The Operating System detected a colon (:) at the start of the pathname specified by the name\$p parameter. This indicates that the pathname contains a logical name. However, the Extended I/O System was unable to find a second colon to terminate the logical name.

E\$PREPOSITION

Someone used a zero as the preposition value. This indicated that the same preposition was to be used as in the last C\$GET\$OUTPUT\$PATHNAME call. Unfortunately, C\$GET\$OUTPUT\$PATHNAME has not been called.

E\$SHARE

When your task invoked this system call, the call invoked at least one other call. While the Operating System performed the latter call, one of the following situations occurred.

- While attempting to open a connection so that your task could access the file through the connection, the Operating System detected the following: Some other task in your system is using the I/O System to manipulate the file through another connection. That task requested that the I/O System restrict the sharing of the file to certain modes. Your task attempted to use a mode that precludes sharing the file.
- While attempting to open a connection so that your task could access the file through the connection, the Operating System detected your task attempting to open a directory for writing only.
- While processing a file-creating system call, your task attempted to create a file that already exists. Consequently, the Extended I/O System must truncate the file to zero length. However, the Extended I/O System cannot do this because, when the file was initially created, the owner specified that it could not be shared with writers.

EXCEPTION CODES

E\$SPACE

When your task invoked this system call, the call invoked at least one other call. While the Operating System performed the latter call, one of the following situations occurred.

- While attempting to create a file or write into a file, the Operating System detected that there was no more space on the volume.
- While attempting to create a file, the Operating System detected that the Extended I/O System had run out of fnodes on the volume. Refer to the FORMAT command in this manual.

E\$SUPPORT

Either the %NO_CREATE_FALSE or the %NO_TRUNCATE macro call was configured into the Basic I/O subsystem.

E\$TIME

Your task used a Human Interface call but was not within a job created by the Human Interface subsystem.

C\$GET\$OUTPUT\$PATHNAME

C\$GET\$OUTPUT\$PATHNAME parses the command line and returns a pathname that refers to the Standard Output file or device.

```
preposition = RQ$C$GET$OUTPUT$PATHNAME(path$name$p, path$name$max,
                                         default$output$p, except$ptr);
```

INPUT PARAMETERS

path\$name\$max A WORD that indicates the maximum size (in bytes) of the path\$name\$ string. Since the maximum size of the buffer is 256 bytes, the maximum allowable size for individual pathnames is 255 characters plus one extra byte.

default\$output\$p A POINTER to a STRING containing the command's default Standard Output. If a TO/OVER/AFTER preposition is not encountered in the command line being processed, the text of this parameter will be used as though it had appeared on the command line. The text must specify TO, OVER, or AFTER for the output mode. Examples: TO :CO: or TO :LP:.

OUTPUT PARAMETERS

preposition A BYTE describing the preposition type that C\$GET\$OUTPUT\$PATHNAME encountered. This value may be passed to C\$GET\$OUTPUT\$CONNECTION if a connection to the file is desired. The value will be one of the following:

<u>Value</u>	<u>Meaning</u>
1	TO
2	OVER
3	AFTER

path\$name\$p A POINTER to a buffer that will receive a STRING containing the next pathname in the Standard Output. A null (zero-length) name will be returned if no Standard Output pathnames were given and no default\$output parameter was specified (a zero pointer or null string).

except\$ptr A POINTER to a WORD in which the Human Interface will return an exception code.

C\$GET\$OUTPUT\$PATHNAME

DESCRIPTION

The pathname returned by C\$GET\$OUTPUT\$PATHNAME may be used for the following purposes:

- In calls to the Extended I/O System. If the Extended I/O System is used, the interpretation of the TO/OVER/AFTER prepositions will be the user's responsibility.
- In a call to C\$GET\$OUTPUT\$CONNECTION to obtain a connection.

On the command line, the Standard Output is denoted by the appearance of the TO/OVER/ AFTER preposition. See "Parameter Parsing" in Chapter 5.

Further calls to C\$GET\$OUTPUT\$PATHNAME will return additional pathnames, either to the next file in the output list, or to the same pathname for a concatenation operation.

EXCEPTION CODES

E\$OK	No exceptional conditions were encountered.
E\$CONTEXT	The Operating System detected a zero value for the object directory size. This indicates that your task used a Human Interface call but was not within a job created by the Human Interface subsystem.
E\$DEFAULT\$SO	The default output name STRING is invalid for one or more of the following reasons: <ul style="list-style-type: none">● The preposition given is not TO, OVER, or AFTER.● The parameter default\$output\$p does not contain a pathname.● Someone did not specify an output preposition on the command line and a default was not given.
E\$LIMIT	The calling job's object limit was exceeded while the objects needed by this call were being created.
E\$LIST	One of the following conditions exist: <ul style="list-style-type: none">● The input pathnames contain unmatched parentheses.● The last value of input pathname list is missing. For example, ABLE,BAKER, has no value after the second comma.
E\$MEM	The memory pool of the job containing the task which invoked this call does not currently have a block of memory large enough to allow this system call to run to completion.

EXCEPTION CODES (continued)

E\$NOT\$CONFIGURED One or both of the following Nucleus System calls were not incorporated during system configuration:

CREATE\$SEGMENT
LOOKUP\$OBJECT

E\$PARSE\$TABLES The Human Interface subsystem detected an error that should not occur unless someone alters an internal table used by this subsystem.

E\$STRING The pathname to be returned exceeds the length limit of 255 characters.

E\$STRING\$BUFFER Someone created a buffer to which the parameter path\$name\$p points. This buffer was not large enough for the pathname to be returned.

E\$TIME Your task used a Human Interface call but was not within a job created by the Human Interface subsystem.

C\$GET\$PARAMETER

GET\$PARAMETER parses the command line to find one parameter and returns it as a keyword name and a list of values.

```
more = RQ$C$GET$PARAMETER(name$p, name$max, value$p, value$max,
                           index$p, predict$list$p, except$ptr);
```

INPUT PARAMETERS

name\$max A WORD that contains the length of the user-supplied buffer (in bytes) specified by name\$p.

value\$max A WORD that contains the length of the user-provided buffer specified in value\$p.

predict\$list\$p A POINTER to a STRING\$TABLE, as described in Appendix C, that contains the predicted prepositions. The predict\$list\$p POINTER should be zero if the preposition prediction function is not being used for the command.

OUTPUT PARAMETERS

more A BYTE value that indicates whether or not the current call to C\$GET\$PARAMETER returned a parameter. A value of 00h indicates there are no more parameters; a value of 0FFh indicates that a parameter was returned.

name\$p A POINTER to the buffer that receives the parameter keyword STRING. If a keyword did not exist for the parameter, a null (zero-length) string will be returned. The size of the buffer must be one byte longer than the largest expected parameter keyword or preposition.

value\$p A POINTER to a buffer that will receive a STRING\$TABLE, as described in Appendix C. The value-part will be stored as a STRING\$TABLE in the buffer. If the parameter had multiple values, the STRING\$TABLE entry would return one value per string.

OUTPUT PARAMETERS (continued)

index\$p	A POINTER to a BYTE that receives the index to the predict\$list\$p entry that corresponds to the name\$p contents. If predict\$list\$p is empty or if the parameter name is not found in the predict list, the index will be zero; that is, the index will be non-zero only if a preposition is found.
except\$ptr	A POINTER to a WORD in which the Human Interface will return an exception code.

DESCRIPTION

C\$GET\$PARAMETER parses one parameter and returns it as a keyword name and a value. A parameter may be one of the following:

- keyword parameter using parenthesis
- keyword parameter using equal sign
- preposition parameter
- list of values

A description of the types, format, and syntax of acceptable parameters is provided in Chapter 5 under "Parameter Parsing."

A STRING\$TABLE is the structure used to implement the predict\$list table. Any parameter you define in the predict\$list table must be used as a preposition. You will also need to provide a STRING\$TABLE entry containing the prepositions that C\$GET\$PARAMETER may encounter; that is, unless you provide some external semantics in a STRING\$TABLE, the parsing routines have no way of distinguishing a preposition parameter from two keyword parameters. See Appendix C for a description and format of a STRING\$TABLE.

EXCEPTION CODES

E\$OK	No exceptional conditions were encountered.
E\$CONTEXT	The Operating System detected a zero value for the object directory size. This indicates that your task used a Human Interface call but was not within a job created by the Human Interface subsystem.
E\$CONTINUED	The Operating System detected a continuation character in the parse buffer while performing the system call. This condition should only occur while parsing the contents of a buffer other than the command line buffer.

C\$GET\$PARAMETER

EXCEPTION CODES (continued)

E\$LIST

One of the following conditions exist:

- The parameter contains unmatched parenthesis.
- A value in the value list is missing or an improper value was entered. Examples of both these conditions follow:

<u>Value</u>	<u>Comments</u>
A,B,	No value following second comma.
A, B,C	Space between first comma and B.
A,B=C,D	The equal sign can not be used unless it is between quotes: 'B=C' is proper.
A,B(C,E),F	The parentheses can not be used unless between quotes.

E\$LITERAL

The Operating System detected a literal (quoted string) in the parse buffer with no closing quote. This condition should only occur while parsing the contents of a buffer other than the command line buffer.

E\$NOT\$CONFIGURED

The Nucleus System call LOOKUP\$OBJECT was not incorporated during system configuration.

E\$PARSE\$TABLES

The Human Interface subsystem has detected an error that should not occur unless someone inadvertently alters an internal table used by this subsystem.

E\$SEPARATOR

The Operating System detected a command separator in the parse buffer while performing this system call. This condition should only occur while parsing the contents of a buffer other than the command line buffer. The following is a list of the command separators: ><, <>, ||, |, [, and].

E\$STRING

One or more of the following conditions exist:

- The string to be returned as the parameter name exceeds the length limit of 255 characters.
- One of the parameter values to be returned exceeds 255 characters in length.

E\$STRING\$BUFFER

One or more of the following conditions exist:

- The string to be returned as the parameter name exceeds the buffer size provided by the user in the call.

EXCEPTION CODES

E\$STRING\$BUFFER (continued)

- The parameter values to be returned exceed the value-buffer size provided by the user in the call.

E\$TIME

Your task used a Human Interface call but was not within a job created by the Human Interface subsystem.

C\$SEND\$COMMAND

C\$SEND\$COMMAND, a command processing call, accepts command lines as read from the user's console and concatenates them into the data space created by the C\$CREATE\$COMMAND\$CONNECTION call.

```
CALL RQ$C$SEND$COMMAND(command$conn, line$p, command$except$ptr,
                        except$ptr);
```

INPUT PARAMETERS

command\$conn A WORD containing a TOKEN for a command connection, where the connection was created by a C\$CREATE\$COMMAND\$CONNECTION call.

line\$p A POINTER to a STRING containing a command to execute.

OUTPUT PARAMETERS

command\$except\$ptr A POINTER to a WORD that receives the status of the invoked command while it is being executed. This parameter is undefined if an exceptional condition code is returned in the exception pointer.

except\$ptr A POINTER to a WORD in which the Human Interface will return the exception code for an exception that occurred while the command was being loaded.

DESCRIPTION

C\$SEND\$COMMAND accepts command lines and combines them into the data space created by the C\$CREATE\$COMMAND\$CONNECTION call. The general structure of a command line is as follows:

```
command-pathname inpath-list [preposition outpath-list] [parameters]
```

The **command-pathname** contains the pathname of the object file of the program that is to be loaded by the Application Loader.

As described in greater detail in this manual's chapter on command entry and syntax, a command line entry may contain several continuation marks. The continuation mark indicates that the command line is continued on the next line. If the command line is continued on another line, an E\$CONTINUED exception code value is returned, the command is not executed, and control returns to the caller.

DESCRIPTION (continued)

If a scan of the command line is successful, the command-pathname entered in the line\$p parameter is parsed. If no exception conditions halt the process at this point, the Human Interface subsystem requests the Application Loader to execute the command.

An Application Loader call creates an I/O job and then the Application Loader validates the header, group definition and segment definition records of the object file of the program to be executed. Refer to the 8086 FAMILY UTILITIES USER'S GUIDE for explanations of segments, groups and object file formats.

C\$SEND\$COMMAND returns two exception codes: one for the C\$SEND\$COMMAND call and one for the invoked command. The except\$ptr parameter returns the C\$SEND\$COMMAND exceptional conditions, as described under the EXCEPTION CODES heading in this command description. The command's except\$ptr returns the invoked command's exceptional condition codes, and these are defined by the invoked command. The E\$CONTROL\$C exception code can be received at either place.

EXCEPTION CODES

E\$OK	No exceptional conditions were encountered.
E\$BAD\$GROUP	The object file represented by the command-pathname contained an invalid group definition record.
E\$BAD\$HEADER	The object file represented by the command-pathname does not begin with a header record for a loadable object module.
E\$BAD\$SEGMENT	The object file represented by the command-pathname contained an invalid segment definition record.
E\$CHECKSUM	At least one record of the object file represented by the command-pathname contains a checksum error. This situation could occur if the CHECKSUM amount calculated during the read operation did not match the CHECKSUM field of the record being read.
E\$CONTEXT	When your task invoked this system call, the Human Interface call invoked at least one other system call. While the Operating System performed the latter call, one of the following situations occurred. <ul style="list-style-type: none"> • The Operating System detected a zero value for the object directory size. This indicated that your task was not within a job created by the Human Interface subsystem.

EXCEPTION CODES

E\$CONTEXT (continued)

- The Operating System detected the device containing the object file of the command-pathname was in the process of being detached.
- The Operating System detected a situation where the calling task's job was not created by the Human Interface subsystem.
- The Extended I/O System was unable to attach the device containing the object file represented by the command-pathname because the Basic I/O System has already attached the device.
- The Operating System detected a command-pathname that refers to a device rather than to a named file.
- The Operating System detected that the UNCATALOG\$OBJECT system call was not incorporated into your system during the configuration process.

E\$CONTINUED

The Operating System detected a continuation character while scanning the command line pointed to by the line\$p parameter. This condition should occur if the command line is to continue on the next line.

E\$CONTROL\$C

The user typed CONTROL-C while the command was being loaded.

E\$DEVFD

Your task forced the Extended I/O System to attempt the physical attachment of a device that had formerly been only logically attached. In the process of attempting to physically attach the device, the Extended I/O System found that the device and the device driver specified in the logical attachment were incompatible. The Operating System would not have returned this exception code if the device referenced by the line\$p parameter had been properly configured in the Extended I/O and/or Basic I/O subsystems.

E\$EOF

The Application Loader returned this exception code because the it encountered an unexpected end of file on the object file represented by the command-pathname.

EXCEPTION CODES (continued)

E\$EXIST	The Operating System detached the device containing the object file represented by the command-pathname before it completed the loading operation.
E\$FACCESS	The Operating System detected the user as not having READ access to the object file represented by the command-pathname.
E\$FIXUP	When the Application Loader load an LTL program, the Loader must adjust some of the addresses used in the code to reflect actual loaded code addresses. This adjustment is known as a fixup and is contained on a fixup record. The Application Loader subsystem detected an invalid fixup record on the object file represented by the command-pathname. Refer to the IRMX 86 LOADER REFERENCE MANUAL for an explanation of Load-time Locatable Code (LTL).
E\$FLUSHING	The Operating System detected the device containing the object file represented by the command-pathname was in the process of being detached.
E\$FNEXIST	Some file in the command-pathname is either marked for deletion or does not exist. For example, the pathname A/B/C assumes that A and B are names for directories and C is the name of an object file. This exception code would have been returned if A, B, or C was marked for deletion or did not exist.
E\$FTYPE	The Operating System detected an error in the command-pathname. The pathname included the name of a data file as a directory. For example, the pathname A/B/C assumes that A and B are names for directories. This exception code would have been returned if either A or B was something other than a directory.
E\$ILLVOL	Your task forced the Extended I/O System to attempt the physical attachment of a device that had formerly been only logically attached. In the process of attempting to physically attach the device, the Extended I/O System examined the volume label and found that the volume does not contain named files. This prevented the Extended I/O System from completing physical attachment because the named file driver was requested during logical attachment.

EXCEPTION CODES (continued)

E\$IO	The Operating System detected an I/O error as it tried to load the object file represented by the command-pathname.
E\$IOMEM	The Basic I/O subsystem job does not currently have a block of memory large enough to allow the Human Interface to create the connection necessary to allow this call to run to completion.
E\$JOB\$PARAM	During the process of configuring the Human Interface subsystem, someone modified either the MIN\$MEMORY or MAX\$MEMORY parameters in the Human Interface configuration. While processing your task, the Operating System detected that the MAX\$MEMORY parameter is both nonzero and smaller than the MIN\$MEMORY.
E\$JOB\$SIZE	During the process of configuring the Human Interface subsystem, someone modified either the MIN\$MEMORY or MAX\$MEMORY parameters of the Human Interface configuration. While processing your task, the Operating System detected that the MAX\$MEMORY parameter is nonzero and too small for the object file represented by the command-pathname to be loaded.
E\$LIMIT	<p>When your task invoked this system call, the call invoked at least one other system call. While the Operating System performed the latter call, one of the following situations occurred.</p> <ul style="list-style-type: none"> • The Human Interface subsystem created enough objects to exceed the object limit of the Basic I/O System job. Refer to the chapter of the iRMX 86 CONFIGURATION GUIDE that discusses the Basic I/O System. • While creating the objects needed by this call, the Operating System detected the calling task's job already having reached the maximum number of objects that can exist simultaneously. • During the process of configuring your application system, the Basic I/O System job was configured a maximum priority that is too low. Specifically, the BIOS maximum priority is lower than either the DUIB priority or the DEVINFO priority. Refer to the iRMX 86 CONFIGURATION GUIDE for information regarding the BIOS maximum priority. Refer to the GUIDE TO WRITING DEVICE DRIVERS FOR THE iRMX 86 AND iRMX 88 I/O SYSTEMS for additional information regarding the DUIB and DEVINFO.

EXCEPTION CODES

E\$LIMIT (continued)

- The job containing the task which invoked this call, or the job's default user object, is currently involved in more than 255 (decimal) I/O operations.
- The calling job's object directory is full or the object directory of the created command's job is full.
- The job containing the task which invoked this call was not created by the Human Interface subsystem.
- A value that specified the priority of the loaded task in the new job is greater than the newly created I/O job's maximum priority. This maximum priority was specified during the configuration of the Human Interface subsystem. Refer to the iRMX 86 CONFIGURATION GUIDE for more information.
- The object directory of the newly created I/O job is full. The size of this object directory was specified during the configuration of the Extended I/O System. Refer to the iRMX 86 CONFIGURATION GUIDE for more information.
- Either the newly created I/O job, or its default user, is currently involved in more than 255 (decimal) I/O operations.

E\$LITERAL

The Operating System detected a literal (quoted string) with no closing quote while scanning the contents of the command line pointed to by the line\$p parameter.

E\$LOADER\$SUPPORT

The object file represented by the command-pathname required capabilities not configured into the Application Loader. For example, you might be attempting to load PIC code with a loader configured only for absolute code. Refer to iRMX 86 LOADER REFERENCE MANUAL for an explanation of PIC code.

E\$LOG\$NAME
\$NEXIST

The command-pathname specified in the line\$p parameter contains an explicit logical name but the Extended I/O System was unable to find this name in the object directories of the local job, the global job, and the root job.

EXCEPTION CODES (continued)

E\$MEDIA	The device containing the object file represented by the command-pathname was not online.
E\$MEM	<p>When your task invoked this call, the call invoked at least one other system call. While the Operating System performed the latter call, one of the following situations occurred.</p> <ul style="list-style-type: none"> ● The Operating System detected the memory pool of the calling task's job as not currently having a block of memory large enough to allow this system call to run to completion. ● The Operating System detected that the memory pool of the newly created I/O job does not currently have a block of memory large enough to allow the initial task to start running. ● The Operating System detected that the memory pool of the Basic I/O System job does not currently have a block of memory large enough to allow the Application Loader to run.
E\$NO\$LOADER\$MEM	<p>When your task invoked this system call, the call invoked at least one other system call. The latter call was a system call of the Application Loader. While attempting to perform the Application Loader system call, one of the following situations occurred.</p> <ul style="list-style-type: none"> ● The Operating System detected the memory pool of the newly created I/O job as not currently having a block of memory large enough to allow the Loader to run. ● The Operating System detected the memory pool of the Basic I/O System's job as not currently having a block of memory large enough to allow the Application Loader to run.
E\$NO\$MEM	The Application Loader attempted to load PIC or LTL groups or segments. However, the memory pool of the newly created I/O job does not currently contain a block of memory large enough to accommodate these groups or segments. Refer to the <code>IRMX 86 LOADER REFERENCE MANUAL</code> for an explanation of loading PIC or LTL groups or segments.

EXCEPTION CODES (continued)

E\$NO\$PREFIX	The command-pathname specified in the line\$p parameter of this call contained no explicit prefix (no logical name), so the Extended I/O System attempted to use the default prefix. However, the default prefix is either undefined, or it is not a valid device connection or file connection.
E\$NO\$START	The object file represented by the command-pathname does not specify the entry point for the program being loaded.
E\$NOT\$CONFIGURED	One or more of the following system calls were not incorporated during system configuration:

<u>Call</u>	<u>Subsystem</u>
A\$ATTACH\$FILE	Basic I/O
A\$CLOSE	Basic I/O
A\$LOAD\$I/O\$JOB	Application Loader
A\$OPEN	Basic I/O
A\$PHYSICAL\$ATTACH\$DEVICE	Basic I/O
A\$READ	Basic I/O
A\$SEEK	Basic I/O
A\$SPECIAL	Basic I/O
CATALOG\$OBJECT	Nucleus
CREATE\$COMPOSITE	Nucleus
CREATE\$I/O\$JOB	Extended I/O
CREATE\$MAILBOX	Nucleus
CREATE\$SEGMENT	Nucleus
CREATE\$TASK	Nucleus
DELETE\$COMPOSITE	Nucleus
DELETE\$TASK	Nucleus
DISABLE\$DELETION	Nucleus
ENABLE\$DELETION	Nucleus
EXIT\$I/O\$JOB	Extended I/O
GET\$DEFAULT\$PREFIX	Basic I/O
GET\$TYPE	Nucleus
LOOKUP\$OBJECT	Nucleus
RECEIVE\$CONTROL	Nucleus
RECEIVE\$MESSAGE	Nucleus
S\$ATTACH\$FILE	Extended I/O
S\$CATALOG\$CONNECTION	Extended I/O
SEND\$CONTROL	Nucleus
SEND\$MESSAGE	Nucleus
SET\$INTERRUPT	Nucleus
UNCATALOG\$OBJECT	Nucleus
WAIT\$INTERRUPT	Nucleus

E\$NOT\$PREFIX	The Operating System detected the command-pathname specified in the line\$p parameter containing a logical name. The logical name referred to an object that was neither a device connection nor a file connection.
----------------	---

EXCEPTION CODES (continued)

E\$NO\$USER	The Operating System detected the calling task's job as not having a default user or detected the default user of the calling task's job as not being a user object.
E\$PARAM	<p>When your task invoked this system call, the call invoked at least one other system call. While the Operating System performed the latter call, one of the following situations occurred.</p> <ul style="list-style-type: none"> • The Operating System detected the command-pathname referenced in the line\$p parameter containing a logical name that was either longer than 12 characters or contained invalid characters. • Your task forced the Extended I/O System to attempt the physical attachment of a device referenced by the command-pathname in the line\$p parameter. This device had formerly been only logically attached. In the process of attempting to physically attach the device, the Extended I/O System found that the logical attachment referred to a file driver (named, physical, or stream) that is not configured into your system. Hence the physical attachment is not possible. • The Operating System detected that the object file represented by the command-pathname has a stack smaller than 16 bytes.
E\$PARSE\$TABLES	The Human Interface subsystem has detected an error that should not occur unless someone alters an internal table used by this subsystem.
E\$PREFIX\$SYNTAX	The Operating System detected a colon (:) at the beginning of the command-pathname specified in the line\$p parameter, which indicates that the pathname contains a logical name. The Extended I/O System, however, was unable to find a second colon to terminate the logical name.
E\$REC\$FORMAT	The Operating System detected that at least one record in the object file represented by the command-pathname contains a format error.
E\$REC\$LENGTH	The object file represented by the command-pathname contains a record that is longer than the Loader's maximum record length. The Loader's maximum record length is a parameter specified during the configuration of the Loader. Refer to the iRMX 86 CONFIGURATION GUIDE for details.

EXCEPTION CODES (continued)

E\$REC\$TYPE	<p>The Application Loader detected one of the following situations while attempting to load the object file represented by the command-pathname.</p> <ul style="list-style-type: none"> ● At least one record in the file being loaded is of a type that the Loader cannot process. ● The Loader has encountered records in a sequence that it cannot process.
E\$SEPARATOR	<p>The Operating System detected a command separator in while scanning the command line pointed to by the line\$p parameter. The following is a list of the command separators: ><, <>, , , [, and].</p>
E\$SHARE	<p>An Application Loader system call was attempting to use the object file represented by the command-pathname that is already being used by some other task. However, the Application Loader was unable to share the file.</p>
E\$STRING	<p>The size of the command-pathname specified in the line\$p parameter exceeds the length limit of 255 (decimal) characters.</p>
E\$STRING\$BUFFER	<p>The size of the command-pathname specified in the line\$p parameter exceeds the size of the command name buffer that someone specified during the configuration of the Human Interface.</p>
E\$TIME	<p>Your task was not within a job created by the Human Interface subsystem.</p>
E\$TYPE	<p>The Operating System detected a command connection parameter that refers to an object that is not a command connection.</p>

C\$SEND\$CO\$RESPONSE

C\$SEND\$CO\$RESPONSE, a message processing call, sends a message to :CO: and reads a response from :CI: that was defined at the time the job was created.

```
CALL RQ$C$SEND$CO$RESPONSE(response$p, response$max, message$p,
                             except$ptr);
```

INPUT PARAMETERS

- message\$p** A POINTER to a STRING containing the message to be sent to :CO:.
- response\$max** A WORD that specifies the maximum number of characters that the response\$p string entry may contain. If response\$max is zero, no response from :CI: will be requested and control will return to the caller immediately.

OUTPUT PARAMETERS

- response\$p** A POINTER to a buffer that will receive a STRING containing the user's response from :CI:.
- except\$ptr** A POINTER to a WORD in which the Human Interface will return an exception code.

DESCRIPTION

If a user response is requested to a C\$SEND\$CO\$RESPONSE message, no other output will be displayed until the user enters a line terminator from a program or console keyboard. If response\$p or response\$max is zero, the Human Interface will not wait for input. If response\$p and response\$max are non-zero, the user may choose to ignore the displayed message by entering a line terminator only. The values contained in message\$p and response\$max have the following meaning:

<u>message\$p</u>	<u>response\$max</u>	<u>Action</u>
zero	zero	No I/O will be performed
zero	non-zero	Send no message; wait for input
non-zero	non-zero	Send message, wait for input
non-zero	zero	Send message, don't wait

DESCRIPTION (continued)

The main distinction between C\$SEND\$CO\$RESPONSE and C\$SEND\$EO\$RESPONSE calls is that the messages sent by C\$SEND\$EO\$RESPONSE are always directed to the user's console screen and cannot be redirected to another device. In contrast, messages output by C\$SEND\$CO\$RESPONSE can be redirected via a SUBMIT command.

EXCEPTION CODES

E\$OK	No exceptional conditions were encountered.
E\$CONTEXT	<p>When your task invoked this system call, the call invoked at least one other system call. While the Operating System performed the latter call, one of the following situations occurred.</p> <ul style="list-style-type: none"> • The Operating System detected a zero value for the object directory size. This indicated that your task used a Human Interface call but was not within a job created by the Human Interface subsystem. • While reading from a file to a buffer, the connection to the file was not open for reading or for both reading and writing. • While reading from a file to a buffer, the connection to the file was closed. • While creating the command connection that created the calling task's job, the CI and CO connections were opened with A\$OPEN rather than S\$OPEN.
E\$IO	The Operating System detected an I/O error during the reading or writing operation.
E\$LIMIT	<p>When your task invoked this system call, the call invoked at least one other system call. While the Operating System performed the latter call, one of the following situations occurred.</p> <ul style="list-style-type: none"> • While creating the objects needed by this call, the Operating System detected the calling task's job having already reached the maximum number of objects that can exist simultaneously. • The job containing the task which invoked this call, or the job's default user object, is currently involved in more than 255 (decimal) I/O operations.

C\$SEND\$CO\$RESPONSE

EXCEPTION CODES

E\$LIMIT (continued)

- The job containing the task which invoked this call was not created by the Human Interface subsystem.

E\$MEM The memory pool of the job containing the task which invoked this call does not currently have block of memory large enough to allow this system call to run to completion.

E\$NOT\$CONFIGURED One or more of the following system calls were not incorporated during system configuration:

<u>Call</u>	<u>Subsystem</u>
A\$READ	Basic I/O
A\$SEEK	Basic I/O
A\$WRITE	Basic I/O
CREATE\$MAILBOX	Nucleus
CREATE\$SEGMENT	Nucleus
DISABLE\$DELETION	Nucleus
ENABLE\$DELETION	Nucleus
GET\$TYPE	Nucleus
LOOKUP\$OBJECT	Nucleus
RECEIVE\$CONTROL	Nucleus
RECEIVE\$MESSAGE	Nucleus
SEND\$CONTROL	Nucleus
SEND\$MESSAGE	Nucleus
S\$READ\$MOVE	Extended I/O
S\$WRITE\$MOVE	Extended I/O

E\$PARAM While processing a write call, the Operating System detected an attempt to write beyond the end of a physical file.

E\$SPACE While processing a write call, the Operating System detected a full volume. The Extended I/O System, however, was unable to complete the requested writing operation.

E\$TIME Your task was not within a job created by the Human Interface subsystem.

C\$SEND\$EO\$RESPONSE

C\$SEND\$EO\$RESPONSE, a message processing call, sends an exceptional condition message to the user's console screen and optionally reads a response from the console keyboard.

```
CALL RQ$C$SEND$EO$RESPONSE(response$p, response$max, message$p,
                             except$ptr);
```

INPUT PARAMETERS

message\$p A POINTER to a STRING containing the message to be sent to the user's console screen. If zero, no message is sent.

response\$max A WORD that gives the maximum number of characters that the response\$p entry may contain. If response\$max is zero, no response from the user's keyboard will be requested and control will return to the caller immediately.

OUTPUT PARAMETERS

response\$p A POINTER to a buffer that receives the STRING containing the user's response from the console keyboard.

except\$ptr A POINTER to a WORD in which the Human Interface will return an exception code.

DESCRIPTION

After C\$SEND\$EO\$RESPONSE has sent a message to the console screen, no other output will be displayed on the screen between the time the message is displayed and a line terminator is typed on the console keyboard. The values contained in message\$p and response\$max have the following meaning:

<u>message\$p</u>	<u>response\$max</u>	<u>Action</u>
zero	zero	No I/O will be performed
zero	non-zero	Send no message; wait for input
non-zero	non-zero	Send message, wait for input
non-zero	zero	Send message, don't wait

C\$SEND\$EO\$RESPONSE

DESCRIPTION (continued)

The main distinction between the C\$SEND\$EO\$RESPONSE and C\$SEND\$CO\$RESPONSE calls is that the messages output by C\$SEND\$EO\$RESPONSE are always sent to the user's console screen and cannot be redirected to another device. In contrast, messages output by C\$SEND\$CO\$RESPONSE can be redirected via a SUBMIT command.

EXCEPTION CODES

E\$OK	No exceptional conditions were encountered.
E\$CONTEXT	<p>When your task invoked this system call, the call invoked at least one other system call. While the Operating System performed the latter call, one of the following situations occurred.</p> <ul style="list-style-type: none">• The Operating System detected a zero value for the object directory size. This indicated that your task used a Human Interface call but was not within a job created by the Human Interface subsystem.• The connection to the file was not open for reading or for both reading and writing.• While reading from a file to a buffer or writing from a buffer to a file, the connection to the file was closed.• While creating the command connection that created the calling task's job, the CI and CO connections were opened with A\$OPEN rather than S\$OPEN.
E\$I/O	In performing a read or write operation, the Operating System detected an I/O error.
E\$LIMIT	<p>When your task invoked this system call, the call invoked at least one other system call. While the Operating System performed the latter call invoked, one of the following situations occurred.</p> <ul style="list-style-type: none">• While creating the objects needed by this call, the Operating System detected the calling task's job having already reached the maximum number of objects that can exist simultaneously.• The job containing the task which invoked this call, or the job's default user object, is currently involved in more than 255 (decimal) I/O operations.

EXCEPTION CODES

E\$LIMIT (continued)

- The job containing the task which invoked this call was not created by the Human Interface subsystem.

E\$MEM The memory pool of the calling task's job as not currently having a block of memory large enough to allow this system call to run to completion.

E\$NOT\$CONFIGURED One or more of the following system calls were not incorporated during system configuration:

<u>Call</u>	<u>Subsystem</u>
A\$READ	Basic I/O
A\$SEEK	Basic I/O
A\$WRITE	Basic I/O
CREATE\$MAILBOX	Nucleus
CREATE\$SEGMENT	Nucleus
DISABLE\$DELETION	Nucleus
ENABLE\$DELETION	Nucleus
GET\$TYPE	Nucleus
LOOKUP\$OBJECT	Nucleus
RECEIVE\$CONTROL	Nucleus
RECEIVE\$MESSAGE	Nucleus
SEND\$CONTROL	Nucleus
SEND\$MESSAGE	Nucleus
S\$READ\$MOVE	Extended I/O
S\$WRITE\$MOVE	Extended

E\$PARAM While trying to write to the output, the Operating System detected an attempt to write beyond the end of a physical file.

E\$SPACE While trying to write to the output, the Operating System detected a full volume. Due to this condition, the requested writing operation has not been completed.

E\$TIME Your task used a Human Interface call but was not within a job created by the Human Interface subsystem.

C\$SET\$CONTROL\$C

C\$SET\$CONTROL\$C, a program control call, changes a calling program's CONTROL C exchange to the semaphore specified in the **C\$SET\$CONTROL\$C** call.

```
CALL RQ$C$SET$CONTROL$C(control$c$semaphore, except$ptr);
```

INPUT PARAMETER

control\$c\$semaphore A WORD containing the TOKEN for the user's semaphore to which a single unit will be sent when a CTRL/c is typed on the console keyboard. If the **control\$c\$semaphore** TOKEN is zero, the program's default CONTROL C semaphore will receive any future CONTROL C units.

OUTPUT PARAMETER

except\$ptr A POINTER to a WORD in which the Human Interface will return an exception code.

DESCRIPTION

This call lets you modify the "standard" response to a CTRL/c entry to a response that is appropriate for the given command. The default action is to delete the command entry.

One unit will be sent to the semaphore each time CTRL/c is typed. Any units sent to the semaphore that exceeds the maximum number specified during system configuration will be ignored.

EXCEPTION CODES

E\$OK No exceptional conditions were encountered.

E\$CONTEXT When your task invoked this system call, the Operating System detected a zero value for the object directory size. This indicated that your task used a Human Interface call but was not within a job created by the Human Interface subsystem. The other possibility is that RQ\$UNCATALOG\$OBJECT was not configured.

E\$NOT\$CONFIGURED The Nucleus System call CATALOG\$OBJECT was not incorporated during system configuration.

EXCEPTION CODES (continued)

E\$LIMIT	When your task invoked this system call, the Operating System detected an object directory that had already reached the maximum object directory size.
E\$TYPE	The TOKEN given in the parameter control\$c\$semaphore is not a TOKEN for a semaphore.

C\$SET\$PARSE\$BUFFER

C\$SET\$PARSE\$BUFFER, a command parsing call, permits parsing the contents of a buffer other than the command line buffer whenever the parsing calls are used.

```
offset = RQ$C$SET$PARSE$BUFFER(buff$p, buff$max, except$ptr);
```

INPUT PARAMETERS

- | | |
|-----------|--|
| buff\$p | A POINTER to a STRING containing the text to be parsed. If the buff\$p is zero, the buffer used for parsing reverts to the command line buffer and the buff\$max parameter is ignored. |
| buff\$max | A WORD that specifies the number of characters in the buffer that are to be used in parsing. |

OUTPUT PARAMETERS

- | | |
|-------------|---|
| offset | A WORD that receives the byte offset, in the previous buffer, of the last byte parsed by the Human Interface library calls. |
| except\$ptr | A POINTER to a WORD in which the Human Interface will return an exception code. |

DESCRIPTION

The user may change buffers and revert back to the command line parsing buffer at will, by calling C\$SET\$PARSE\$BUFFER with buff\$p=0. However, only one parsing buffer per job may be active at any given time.

Note that C\$SET\$PARSE\$BUFFER does not affect C\$GET\$INPUT\$PATHNAME or C\$GET\$OUTPUT\$PATHNAME calls; that is, C\$SET\$PARSE\$BUFFER does not redirect these calls' input source.

EXCEPTION CODES

- | | |
|-------|---|
| E\$OK | No exceptional conditions were encountered. |
|-------|---|

EXCEPTION CODES (continued)

E\$CONTEXT	When your task invoked this system call, the Operating System detected a zero value for the object directory size. This indicated that your task was not within a job created by the Human Interface subsystem.
E\$LIMIT	While processing this calling jobs object directory, the Operating System detected an object directory that has already reached the maximum object directory size.
E\$NOT\$CONFIGURED	The Nucleus System call LOOKUP\$OBJECT was not incorporated during system configuration.
E\$TIME	Your task was not within a job created by the Human Interface subsystem.

CHAPTER 7. COMMAND CREATION EXAMPLES

The examples provided in this chapter are intended to provide some insight as to how and why Human Interface system calls are used to perform command parsing and processing in user-designed programs. The examples presented in following pages are not complete programs and are not tested. Therefore, they should be used only for general guidance when preparing your own commands.

EXAMPLE 1. STANDARD SYSTEM CALLS

Shows implementation of the following "standard" Human Interface system calls:

```
C$GET$INPUT$PATHNAME
C$GET$OUTPUT$PATHNAME
C$GET$INPUT$CONNECTION
C$GET$OUTPUT$CONNECTION
```

EXAMPLE 2. PARSING CALL

Shows implementation of the following Human Interface parsing call:

```
C$GET$PARAMETER
```

EXAMPLE 3. ADVANCED SYSTEM CALLS

Show implementation of the following "advanced" Human Interface system calls:

```
C$CREATE$COMMAND$CONNECTION
C$SEND$COMMAND
C$DELETE$COMMAND$CONNECTION
```

COMMAND CREATION EXAMPLES

EXAMPLE 1. STANDARD SYSTEM CALLS

```

/*****
* This example demonstrates the use of the following Human Interface
* standard input and output functions.
*
*      rq$$get$input$pathname
*      rq$$get$output$pathname
*      rq$$get$input$connection
*      rq$$get$output$connection
*
* This program is a possible implementation of a copy utility, the
* purpose of which is to copy data from successive input files to the
* corresponding successive output files. For example, to copy file A
* to file B, file C to file D, and file E to file F, we could specify
* in a command line:
*      COPY A,C,E TO B,D,F
*
*****/

copy: DO;

DECLARE (input$pathname, output$pathname) structure (
                                length byte,
                                string (41) byte ),
        output$prep byte,
        (input$token, output$token) word,
        status word;

/* Get the 1st input pathname string */
CALL rq$$get$input$pathname(@input$pathname, SIZE(input$pathname), @status);

DO WHILE (input$pathname.length <> 0); /* A zero length indicates no more
                                        input parameters. */

    /* Get the corresponding output pathname string */
    output$prep = rq$$get$output$pathname(@output$pathname,
                                        SIZE(output$pathname), @(7,'TO :CO:'),
                                        @status);

    /* Establish connection with next pair of input and output files */
    input$token = rq$$get$input$connection(@input$pathname, @status);
    output$token = rq$$get$output$connection(@output$pathname,
                                        output$prep, @status);

        .
        . Copy data and close both files.
        .

    /* Get the next input pathname string */
    CALL rq$$get$input$pathname(@input$pathname, SIZE(input$pathname), @status);

END /* DO WHILE */;

END copy;

```

COMMAND CREATION EXAMPLES

EXAMPLE 2. PARSING CALL

```

/*****
*
* This example demonstrates the use of the following Human Interface
* parse standard function:
*
*      rq$c$get$parameter
*
* This program makes use of the get$parameter primitive to parse out a
* keyword parameter in a command line. Here, the keyword, "SIZE", is
* parsed out, and its value-part converted to a word value and placed
* "size$val". For example, we could specify in a command line:
*
*          PROGL    SIZE = 400
*
* Note that if the "SIZE" parameter is not present, "size$val" receives
* a default value.
*
*****/

progl: DO;

DECLARE STRING LITERALLY 'STRUCTURE (len BYTE, str (1) BYTE)',
          STRING$TABLE LITERALLY 'STRUCTURE (num$entries BYTE,
                                     entries (1) BYTE)',
          PARAMETER$KEYWORD$MAX LITERALLY '20',
          VALUE$TABLE$MAX LITERALLY '30',
          DEFAULT$SIZE LITERALLY '100';
DECLARE value$table$buf (VALUE$TABLE$MAX) BYTE, /* Receives value string tbl.*/
        value$table STRING$TABLE AT (@value$table$buf),
        value$str$ptr POINTER,
        value$str based value$str$ptr STRING, /* For referencing strings
                                                within the string table */
        parameter$keyword$buf (PARAMETER$KEYWORD$MAX) BYTE, /* Receives the
                                                                keyword string*/
        parameter$keyword STRING AT (@parameter$keyword$buf),
        predict$list (*) BYTE DATA (2,2,'TO',5,'AFTER'), /* Sample predict
                                                            list table */
        predict$list$index BYTE, /* Receives prep. string displacement within
                                   predict$list, if a preposition is present*/
        status WORD,
        (size$val, i) WORD;

/* Get the next parameter, if present */
IF ( rq$c$get$parameter (@parameter$keyword, PARAMETER$KEYWORD$MAX,
                        @value$table, VALUE$TABLE$MAX,
                        @predict$list$index, @predict$list,
                        @status ) ) THEN
  IF (parameter$keyword.str(0) = 'S') AND /* Is the keyword 'SIZE' ? */
      (parameter$keyword.str(1) = 'I') THEN
    DO;
      value$str$ptr = @value$table.entries; /* Point to 1st entry in table */
      size$val = 0;
      DO i = 0 to value$str.len - 1; /* Convert number string to a word value */
        size$val = size$val * 10;
        size$val = size$val + (value$str.str(i) - 30H);
      END;
    END;
  ELSE
    size$val = DEFAULT$SIZE; /* If the 'SIZE' parameter is not present,
                              use the default size. */

      .
      . Rest of program
      .

END progl;

```

COMMAND CREATION EXAMPLES

EXAMPLE 3. ADVANCED SYSTEM CALLS

```

/*****
*
* This example demonstrates the use of the following Human Interface
* advanced standard functions:
*
*      rq$c$create$command$connection
*      rq$c$send$command
*      rq$c$delete$command$connection
*
* This program uses the above functions to invoke a copy utility
* from within, and then continue normal processing. The program is
* invoked by the command line:
*
*                               PROG2
*
*****/

prog2: DO;

DECLARE (ci$token, co$token, command$connection$token) WORD,
        (status, status1) WORD;

        .
        .
        .

/* INVOKE UTILITY TO COPY FILE OLD TO FILE NEW */

/* Get tokens for CI and CO */
ci$token = rq$c$get$input$connection(@4,':CI:'), @status);
co$token = rq$c$get$output$connection(@4,':CO:'), @status);

/* Create command connection */
command$connection$tok = rq$c$create$command$connection (ci$token,
                                                         co$token,
                                                         Ø,
                                                         @status );

/* Send command to copy files */
CALL rq$c$send$command (command$connection$tok,
                       @23,'COPY :F1:OLD TO :F1:NEW'),
                       @status1, @status);

/* Delete command connection */
CALL rq$c$delete$command$connection (command$connection$tok, @status);

        .
        . Rest of program
        .

END prog2;

```

APPENDIX A. HUMAN INTERFACE TYPE DEFINITIONS

The type definitions used in Human Interface system call description are defined in Table A-1.

Table A-1. Type Definitions

Type	Definition
BYTE	PL/M 86 data type (8 bits).
WORD	PL/M 86 data type (16 bits).
POINTER	PL/M 86 data type (32 bits).
STRING	A sequence of bytes, the first of which contains the number of bytes in the sequence, (not including the count byte). Since the count is only a byte, the maximum number of characters that a STRING may contain is 255 characters. A zero length specifies a null string.
TOKEN	A 2-byte iRMX 86 variable that contains the means of locating an object.

APPENDIX B. HUMAN INTERFACE EXCEPTION CODES

Like other iRMX 86 software systems, the Human Interface returns a condition code whenever a Human Interface call is invoked. If the call executes without error, the Human Interface returns the code E\$OK. When an error is encountered during call execution, an exceptional condition code is returned. The exceptional condition code may be returned either from the Human Interface or from one of the other iRMX 86 systems residing below it.

The exception codes listed in Table B-1 are unique to the Human Interface.

Table B-1. Human Interface Exception Codes

Programmer Errors:
E\$PARSE\$TABLES: 8080h E\$DEFAULT\$SO: 8083h E\$STRING: 8084h
Environmental Errors:
E\$OK: 0000h E\$LITERAL: 0080h E\$STRING\$BUFFER: 0081h E\$SEPARATOR: 0082h E\$CONTINUED: 0083h E\$LIST: 0085h E\$PREPOSITION: 0087h E\$PATH: 0088h E\$CONTROL\$C: 0089h E\$CONTROL: 008Ah E\$EXTRA\$SO 008Bh

HUMAN INTERFACE EXCEPTION CODES

Other exception codes may be issued during Human Interface operations. The hexadecimal values of these exception conditions fall into ranges based on the subsystem which first detects the condition. Table B-2 lists the subsystems and their respective ranges.

Table B-2. Exception Code Ranges

System	Environmental	Programming
Nucleus	0 to 1FH	8000 to 801FH
Basic I/O System	20 to 3FH	8020 to 803FH
Extended I/O System	40 to 5FH	8040 to 805FH
Application Loader	60 to 7FH	8060 to 807FH
Human Interface	80 to AFH	8080 to 80AFH
Universal Development Interface	C0 to DFH	80C0 to 80DFH
Reserved *	130 to 14FH	8130 to 814FH
<p>* Exception codes in this range could occur if you are a user of an iRMX system with MMX software. Refer to iMMX 800 SOFTWARE REFERENCE MANUAL AND USER'S GUIDE for an explanation of exception conditions within this range.</p>		

HUMAN INTERFACE EXCEPTION CODES

Table B-2 provides a minimum of information about an exception condition. In most cases, the exception condition must be considered in terms of the unique circumstances that caused the condition. Table B-3 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 B-3. 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	S	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.
6H	E\$EXIST	* * * * *	A token parameter has a value which is not the token of an existing object.
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		S System Programmer's Ref Manual

HUMAN INTERFACE EXCEPTION CODES

Table B-3. iRMX 86™ Condition Codes

Hex. Value	Mnemonic	Manuals N B E L H	Meaning
Environmental Conditions (continued)			
7H	E\$STATE	*	A task attempted an operation which would have caused an impossible transition of a task's state.
8H	E\$NOT\$CONFIGURED	* * * * *	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.
28H	E\$SHARE	* * * *	Improper file sharing requested.
29H	E\$SPACE	* *	No space left.
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		S System Programmer's Ref Manual	

HUMAN INTERFACE EXCEPTION CODES

Table B-3. iRMX 86™ Condition Codes

Hex. Value	Mnemonic	Manuals N B E L H	Meaning
Environmental Conditions (continued)			
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	S *	Invalidly named volume.
2EH	E\$DEV\$OFF-LINE	*	The device being accessed is now offline.
40H	E\$PREFIX\$-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.
60H	E\$ABS\$ADDRESS	*	An absolute object program was loaded into system protected memory area.
61H	E\$BAD\$GROUP	* *	Illegal group component in the a group definition record.
62H	E\$BAD\$-HEADER	* *	Illegal header 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		S System Programmer's Ref Manual

HUMAN INTERFACE EXCEPTION CODES

Table B-3. iRMX 86™ Condition Codes

Hex. Value	Mnemonic	Manuals N B E L H	Meaning
Environmental Conditions (continued)			
63H	E\$BAD\$SEG- MENT	* *	Illegal 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	* *	Illegal fixup record in the object file.
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\$FMT	* *	Illegal record format encountered.
6AH	E\$REC\$- LENGTH	* *	Record length of an object record exceeds configured loader-buffer size.
6BH	E\$REC\$TYPE	* *	Illegal 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\$OVLY	*	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.
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		S System Programmer's Ref Manual

HUMAN INTERFACE EXCEPTION CODES

Table B-3. iRMX 86™ Condition Codes

Hex. Value	Mnemonic	Manuals N B E L H	Meaning
Environmental Conditions (continued)			
80H	E\$LITERAL	*	The parse buffer contains a literal with no closing quote.
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.
83	E\$CONTINUED	*	The parse buffer contains a continuation character.
85H	E\$LIST	*	The last value of the value list is missing.
87H	E\$PREPOSI- TION	*	The same preposition as on the the command line was indicated, but can not be used.
89H	E\$CONTROL\$C	*	The user typed CONTROL-C while the command was being loaded.
8BH	E\$EXTRA\$SO	*	There were no more input pathnames although the output pathname list was not empty.
Programmer Errors			
8000H	E\$ZERO\$- DIVIDE	*	A task attempted to divide by zero.
8001H	E\$OVER-FLOW	*	An overflow interrupt occurred.
8002H	E\$TYPE	* * * * *	A token parameter referred to an existing object that is not of the required 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		S System Programmer's Ref Manual

HUMAN INTERFACE EXCEPTION CODES

Table B-3. iRMX 86™ Condition Codes

Hex. Value	Mnemonic	Manuals N B E L H	Meaning
Programmer Errors (continued)			
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.
8020H	E\$IFDR	* *	Invalid file driver request.
8021H	E\$NOUSER	* * *	No default user.
8022H	E\$NO\$PREFIX	* * *	No default prefix.
8040H	E\$NOT\$- PREFIX	* *	Specified object is not a device connection or file connection.
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.
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		S System Programmer's Ref Manual	

APPENDIX C. STRING TABLE FORMAT

The string table may be generated by using standard PL/M declarations. The diagram in Figure C-1 shows the string\$table parameter format.

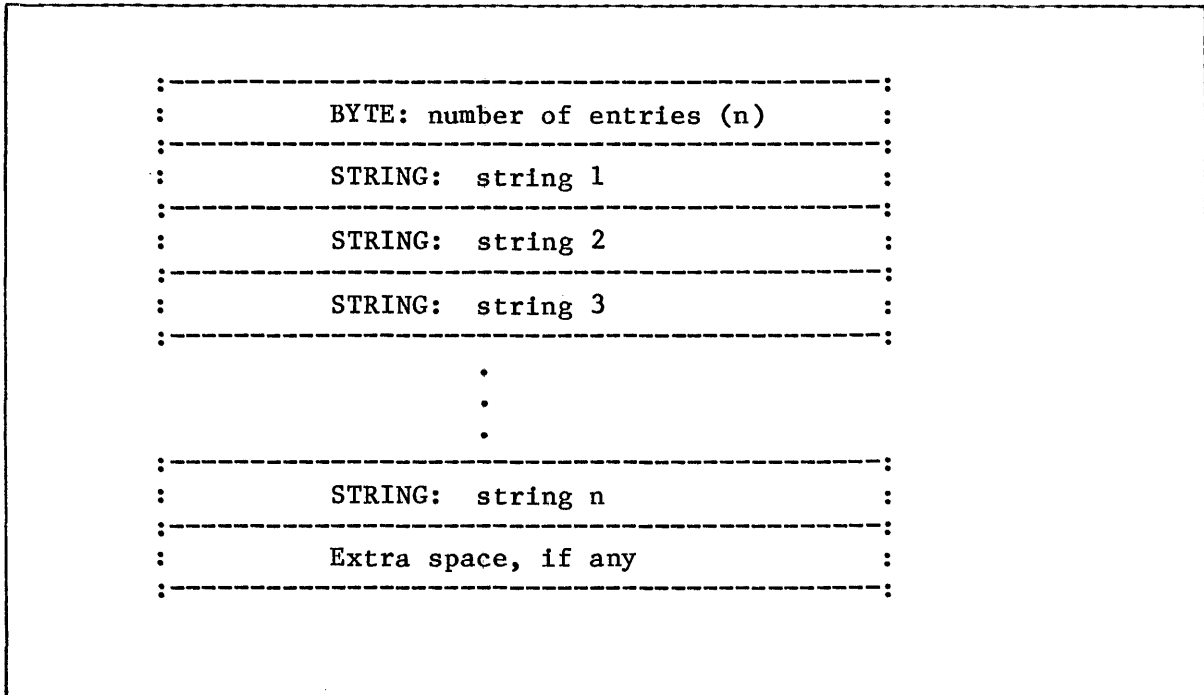


Figure C-1. String Table Format

See Appendix A for a definition of a STRING.

EXAMPLE:

Assume you wish to generate a preposition list for HAPPY, GLAD, and SAD. The following declarations would be needed:

```
DECLARE
  p$table(*) BYTE DATA(3,          /* NUMBER OF STRINGS */
                        (5, 'HAPPY'),
                        (4, 'GLAD'),
                        (3, 'SAD'), );
```

INDEX

Underscored entries are primary references.

:\$: prefix 1-6, 1-7, 1-8
:BB: prefix 1-7, 1-8
:CI: prefix 1-3, 1-7, 1-8, 2-5, 4-2, 4-6, 4-10, 6-4, 6-48
:CO: prefix 1-3, 1-7, 1-8, 2-5, 4-4, 4-7, 4-12, 6-4, 6-14, 6-23, 6-31,
6-48
:PROG: prefix 1-2, 1-5, 1-8, 5-7
:SYSTEM: prefix 1-2, 1-5, 1-8, 5-7
:WORK: prefix 1-6, 1-8

A\$GET\$EXTENSION\$DATA system call 3-42
A\$SET\$EXTENSION\$DATA system call 3-42
absolute load 5-8
access rights 3-31, 3-51
AFTER preposition 2-6, 4-3, 4-6, 4-12, 5-4, 5-7, 6-22, 6-31, 6-51
ampersand 2-3
AS preposition 2-7
ATTACH\$FILE 6-6, 6-7, 6-9, 6-15, 6-18, 6-27, 6-28, 6-45
ATTACHDEVICE command 1-7, 2-7, 3-5, 5-8

BACKUP command 3-9, 3-49
buffer parsing 6-56
BYTE definition A-1

C\$CREATE\$COMMAND\$CONNECTION system call 5-5, 6-3, 6-4, 6-10, 6-38, 7-1
C\$DELETE\$COMMAND\$CONNECTION system call 5-5, 6-3, 6-4, 6-10, 7-1
C\$FORMAT\$EXCEPTION system call 5-8, 6-2, 6-11
C\$GET\$CHAR system call 6-2, 6-13
C\$GET\$EXCEPTION\$HANDLER system call 5-8
C\$GET\$INPUT\$CONNECTION system call 5-3, 6-2, 6-14, 6-15, 6-20, 7-1
C\$GET\$INPUT\$PATHNAME system call 5-1, 5-3, 6-2, 6-20, 6-56, 7-1
C\$GET\$OUTPUT\$CONNECTION system call 5-3, 5-4, 6-2, 6-22, 6-23,
6-31, 6-32, 7-1
C\$GET\$OUTPUT\$PATHNAME system call 6-2, 6-29, 6-31, 6-32, 6-56
C\$GET\$PARAMETER system call 5-1, 5-6, 6-2, 6-34, 6-35, 7-1
C\$SEND\$CO\$RESPONSE system call 6-2, 6-48, 6-49, 6-52
C\$SEND\$COMMAND system call 5-5, 6-3, 6-4, 6-38, 6-39, 7-1
C\$SEND\$EO\$RESPONSE system call 6-2, 6-49, 6-51, 6-52
C\$SET\$CONTROL\$C system call 5-5, 5-8, 6-3, 6-54
C\$SET\$PARSE\$BUFFER system call 6-2, 6-56
CATALOG\$OBJECT system call 6-7, 6-45, 6-54
CLOSE system call 6-45
Command Line Interpreter 1-1, 2-3, 5-4, 5-6

INDEX

command

- connection 5-5, 6-3, 6-4, 6-10
- creation 5-7
- dictionary 3-3
- entry 2-1, 2-4
- examples 4-1
- Human Interface 3-1
- invocation 5-7, 6-4
- line 1-1, 2-2, 2-8, 4-2, 4-7, 4-20, 6-2, 6-13, 6-20, 6-22, 6-31, 6-34, 6-38, 6-56
- name 1-1, 1-4, 2-2, 5-2, 6-47
- parsing 6-2, 6-20, 6-31, 6-34, 6-56, 7-1
- processing calls 5-5, 7-1
- syntax 2-1, 3-1, 2-2

comment character 2-3

console

- input device 2-5
- output device 2-5
- session 2-1

continuation mark 2-3, 6-38

CONTROL keys 2-7

COPY command 2-4, 3-15, 4-2, 4-4, 4-6, 4-12, 4-16, 4-20, 4-21

CREATE\$COMMAND\$CONNECTION system call 5-5, 6-3, 6-4, 6-10, 6-38, 7-1

CREATE\$COMPOSITE system call 6-7, 6-18, 6-27, 6-45

CREATE\$FILE system call 6-7, 6-8, 6-9, 6-27, 6-28

CREATE\$IO\$JOB system call 6-45

CREATE\$MAILBOX system call 6-7, 6-18, 6-27, 6-45, 6-50, 6-53

CREATE\$SEGMENT system call 6-7, 6-18, 6-21, 6-28, 6-33, 6-45, 6-50, 6-53

CREATE\$TASK system call 6-7, 6-45

CREATEDIR command 1-5, 1-6, 3-19, 4-9

CTRL keys

- CTRL key 2-7
- CTRL/c 1-4, 2-7, 4-5, 5-5, 5-8, 6-54
- CTRL/d 2-7
- CTRL/q 2-7, 4-5
- CTRL/s 2-7, 4-5
- CTRL/x 2-7, 4-5
- CTRL/z 2-7, 4-1, 4-2

DATE command 3-20

DEBUG command 3-21

default directory 1-6, 1-8, 4-13

default message 6-2, 6-11

DELETE access 6-23

DELETE command 3-22, 4-8, 4-10, 4-13

DELETE\$COMMAND\$CONNECTION system call 5-5, 6-3, 6-4, 6-10, 7-1

DELETE\$COMPOSITE system call 6-7, 6-18, 6-28, 6-45

DELETE\$TASK system call 6-45

DETACHDEVICE command 3-24

Device Unit Information Block (DUIB) 3-5, 3-21, 6-7, 6-17, 6-26, 6-42

INDEX

device
 attachment 3-5
 detachment 3-25
 fnodes 4-18
 granularity 4-18
 interleave factor 4-18
 logical
 physical
DIR command 1-5, 2-9, 3-25, 4-9, 4-12, 4-18, 4-19
directory 1-4, 1-5, 1-6, 4-2, 4-8, 4-9, 4-10, 4-11, 4-12, 4-13, 4-14,
 4-16, 4-18, 4-19, 5-7
directory
 creation 3-19, 4-9
 deletion 3-22, 4-13
 examples 3-27
 formats 3-29
 listing 3-25, 4-12
 parent 4-10
 renaming 3-45, 4-16
 restoring 3-48
 user default 4-13
DISABLE\$DELETION system call 6-8, 6-18, 6-28, 6-45, 6-50, 6-53
diskette switching 4-19
DISKVERIFY command 3-32
DOWNCOPY command 3-37
DUIB 3-5, 3-21, 6-7, 6-17, 6-26, 6-42

embedded blanks 2-2, 2-3
ENABLE\$DELETION system call 6-8, 6-18, 6-28, 6-45, 6-50, 6-53
error input device 6-2
error messages 2-1
error output device 6-2
ESC key 2-2, 2-3, 2-8, 5-1
exception code 5-8, 6-5, 6-10, 6-12, 6-13, 6-15, 6-20, 6-24, 6-32, 6-35,
 6-39, 6-49, 6-52, 6-54, 6-56
exception code mnemonics,
 E\$CONTEXT 6-5, 6-13, 6-15, 6-21, 6-24, 6-32, 6-35, 6-39, 6-40,
 6-49, 6-52, 6-54, 6-57
 E\$CONTINUED 6-35, 6-38, 6-40
 E\$CONTROL\$C 6-39, 6-40
 E\$DEFAULT\$SO 6-32
 E\$DEVFD 6-5, 6-16, 6-24, 6-40
 E\$EXIST 6-6, 6-41
 E\$EXTRA\$SO 6-21
 E\$FACCESS 6-16, 6-24, 6-25, 6-41
 E\$FLUSHING 6-41
 E\$FNEXIST 6-16, 6-25, 6-41
 E\$FTYPE 6-16, 6-25, 6-41
 E\$ILLVOL 6-16, 6-25, 6-41
 E\$IO 4-19, 6-6, 6-17, 6-25, 6-42, 6-49, 6-52
 E\$IOMEM 6-6, 6-17, 6-25, 6-42
 E\$LIMIT 6-6, 6-7, 6-17, 6-21, 6-25, 6-26, 6-32, 6-42, 6-43, 6-49,
 6-50, 6-52, 6-53, 6-55, 6-57
 E\$LIST 6-21, 6-32, 6-36

INDEX

exception code mnemonics (continued)

E\$LITERAL 6-36, 6-43
E\$MEDIA 6-17, 6-26, 6-44
E\$MEM 6-7, 6-18, 6-21, 6-27, 6-32, 6-44, 6-50, 6-53
E\$NO\$PREFIX 6-18, 6-27, 6-45
E\$NOT\$CONFIGURED 6-7, 6-8, 6-13, 6-18, 6-21, 6-27, 6-28, 6-33,
6-36, 6-45, 6-50, 6-53, 6-54, 6-57
E\$NOT\$PREFIX 6-8, 6-19, 6-28, 6-45
E\$OK 6-5, 6-10, 6-12, 6-13, 6-15, 6-20, 6-24, 6-32, 6-35, 6-39,
6-49, 6-52, 6-54, 6-56
E\$PARAM 4-4, 6-8, 6-12, 6-19, 6-28, 6-29, 6-46, 6-50, 6-53
E\$PARSE\$TABLES 6-21, 6-33, 6-36, 6-46
E\$PREFIX\$SYNTAX 6-19, 6-29, 6-46
E\$PREPOSITION 6-29
E\$SEPARATOR 6-36, 6-47
E\$SHARE 6-19, 6-29, 6-47
E\$STRING 6-12, 6-21, 6-33, 6-36, 6-37, 6-47
E\$STRING\$BUFFER 6-12, 6-21, 6-33, 6-36, 6-37, 6-47
E\$SUPPORT 6-9, 6-30
E\$TIME 6-9, 6-13, 6-19, 6-21, 6-30, 6-33, 6-37, 6-47, 6-50, 6-53,
6-57
E\$TYPE 6-9, 6-10, 6-47, 6-55
Exception handler 5-8
EXIT\$IO\$JOB system call 6-45
Extended I/O System 1-1, 1-2, 5-3, 6-5, 6-7, 6-8, 6-14, 6-15, 6-16,
6-17, 6-18, 6-19, 6-20, 6-22, 6-23, 6-24, 6-25, 6-27, 6-28, 6-29,
6-30, 6-32, 6-40, 6-41, 6-43, 6-45, 6-46, 6-50, 6-53
EXTENDED parameter 2-2
extension data 3-11, 3-41, 3-42, 3-52

file

access rights 3-31
backup 3-8
concatenation 3-15, 4-6
creation 3-15, 4-2
deletion 3-22, 4-8
duplication 3-15, 4-3
granularity 3-31
handling 2-2, 2-5, 2-8, 2-9, 4-1
length 3-31
listing 3-15, 4-4
name 4-2
owner 3-31, 3-49
renaming 3-45, 4-14
replacement 3-15, 4-5
restoring 3-48
scrolling 4-5
sequencing 4-7
structures 1-1, 1-4
SUBMIT 3-55, 4-20
fnodes 3-41, 3-42, 4-18
formal parameters 3-57
FORMAT command 1-7, 3-40, 4-18, 6-30
FORMAT\$EXCEPTION system call 5-8, 6-2, 6-11

INDEX

GET\$CHAR system call 6-2, 6-13
GET\$CONNECTION\$STATUS system call 6-8, 6-28
GET\$DEFAULT\$PREFIX system call 6-8, 6-18, 6-28, 6-45
GET\$EXCEPTION\$HANDLER system call 5-8
GET\$FILE\$STATUS system call 6-7, 6-18, 6-27
GET\$INPUT\$CONNECTION system call 5-3, 6-2, 6-14, 6-15, 6-20, 7-1
GET\$INPUT\$PATHNAME system call 5-1, 5-3, 6-2, 6-20, 6-56, 7-1
GET\$OUTPUT\$CONNECTION system call 5-3, 5-4, 6-2, 6-22, 6-23, 6-31, 6-32,
7-1
GET\$PARAMETER system call 5-1, 5-6, 6-2, 6-34, 6-35, 7-1
GET\$TYPE system call 6-8, 6-18, 6-28, 6-45, 6-50, 6-53
granularity 3-31, 3-41, 3-42, 4-18

Human Interface
 commands 3-1
 services 1-2
 software requirements 1-1
 system calls 1-3, 6-2
 type definitions A-1
I/O processing 5-3, 6-2, 6-14, 6-20, 6-22
input and output calls 5-2
interleave factor 3-41, 4-18
iSBC 957A/B package 3-21, 3-37, 3-59
ISIS-II files 1-3, 3-37, 3-59

keyboard character entries 2-4
keyword parameters 6-35

line terminator 6-48, 6-51
listing formats 1-3
load module formats 5-8
LOAD\$IO\$JOB system call 6-45
logical device 4-19
logical names 1-5, 1-7, 2-5, 5-8
LOOKUP\$OBJECT system call 6-8, 6-13, 6-18, 6-21, 6-28, 6-33, 6-36, 6-45,
6-50, 6-53, 6-57
message processing 5-2, 5-4, 6-2, 6-11, 6-48, 6-51

NAMED files 4-18
named verification 3-33
NEW LINE key 2-3
NO_CREATE_FALSE 6-30
NO_TRUNCATE 6-30

object directory 6-5, 6-6, 6-13, 6-15, 6-21, 6-24, 6-32, 6-35, 6-39,
6-43, 6-49, 6-52, 6-54, 6-55, 6-57
outpath-list 2-2, 6-38
output parameters 2-5, 2-8, 6-4, 6-13, 6-14, 6-20, 6-22, 6-31, 6-34,
6-35, 6-38, 6-48, 6-51, 6-56
OVER preposition 2-6, 2-8, 4-5, 4-15, 5-4, 5-6, 6-22, 6-31, 6-32
owner 3-31, 3-49

parameter 1-1, 2-1, 4-20, 5-1, 5-2, 5-5, 5-9
parameter parsing 5-5, 6-32, 6-35
parent directory 4-10, 6-24

INDEX

parsing calls 6-2, 6-56
 pathlists 2-5
 pathname 1-4, 1-6, 2-2, 2-8, 4-3, 4-9, 4-14, 4-16, 5-7
 PHYSICAL 3-41, 4-18
 physical device 2-5, 2-7, 4-18
 physical verification 3-33
 PHYSICAL\$ATTACH\$DEVICE system call 6-7, 6-18, 6-27, 6-45
 POINTER definition A-1
 predict\$stable 5-6
 preposition 2-2, 5-2, 5-4, 5-5, 6-22, 6-23, 6-29, 6-31, 6-32, 6-34,
 6-35, 6-38
 preposition parameters 2-6, 2-9, 5-5, 5-6
 program control call 5-5, 6-3, 6-54
 program debugging 3-21
 program directories 1-5

 QUERY parameter 2-2, 2-4, 4-3, 4-13
 quoted string 2-3, 5-6, 6-36, 6-43

 READ\$MOVE system call 6-28, 6-50, 6-53
 RECEIVES\$CONTROL system call 6-8, 6-18, 6-28, 6-45, 6-50, 6-53
 RECEIVES\$MESSAGE system call 6-8, 6-18, 6-28, 6-45, 6-50, 6-53
 relocatable load format 5-8, 5-9
 RENAME command 2-8, 2-9, 3-44, 4-5, 4-9, 4-13, 4-14, 4-16
 RESTORE command 3-9, 3-48
 RETURN key 2-2, 2-3, 2-4, 2-7, 2-8, 5-1
 root directory 4-17
 RUBOUT key 2-8

 SEEK system call 6-27, 6-28, 6-45, 6-50, 6-53
 semicolon character 2-3
 SEND\$CO\$RESPONSE system call 6-2, 6-48, 6-49, 6-52
 SEND\$COMMAND system call 5-5, 6-3, 6-4, 6-38, 6-39, 7-1
 SEND\$CONTROL system call 6-8, 6-18, 6-28, 6-45, 6-50, 6-53
 SEND\$EO\$RESPONSE system call 6-2, 6-49, 6-51, 6-52
 SEND\$MESSAGE system call 6-8, 6-18, 6-28, 6-45, 6-48, 6-50, 6-51, 6-53
 SET\$CONTROL\$C system call 5-5, 5-8, 6-3, 6-54
 SET\$EXCEPTION\$HANDLER system call 5-8
 SET\$INTERERRUPT system call 6-8, 6-18, 6-28, 6-45
 SET\$PARSE\$BUFFER system call 6-2, 6-56
 single-stepping 3-21
 slash separator 4-10
 SPECIAL system call 6-7, 6-18, 6-27, 6-45
 Standard Input 5-3, 5-4, 6-2, 6-20
 Standard Output 5-3, 5-4, 6-2, 6-31, 6-32
 STRING 6-1, 6-11, 6-14, 6-20, 6-22, 6-31, 6-48, 6-51, 6-56
 STRING\$TABLE 5-6, 6-34, 6-35
 SUBMIT command 2-3, 3-55, 4-20, 6-49, 6-52
 submit file 4-20
 system
 call dictionary 6-2
 clock 3-58
 date 3-20
 program directory 1-2, 5-7

INDEX

TIME command 3-58
TO preposition 2-6, 4-4, 4-6, 4-12, 4-14, 4-16, 5-4, 6-22, 6-23, 6-31
token definition A-1
TRUNCATE system call 6-27, 6-28
TRUNCATE\$FILE system call 6-28
type definition A-1

UNCATALOG\$OBJECT system call 6-40, 6-45, 6-54
UPCOPY command 3-59, 5-9
User directories 1-6, 4-8

verification utility 3-32
volume
 boundaries 4-16
 formatting 3-40, 4-18
 granularity 3-41, 3-42
 prefix 4-17
 verification 3-32

WAIT\$INTERRUPT system call 6-8, 6-18, 6-28, 6-45
WORD 6-1, 6-4
work directory 1-6
WRITE system call 6-27, 6-28, 6-50, 6-53
WRITE\$MOVE system call 6-28, 6-50, 6-53



REQUEST FOR READER'S COMMENTS

Intel Corporation attempts to provide documents that meet the needs of all Intel product users. This form lets you participate directly in the documentation process.

Please restrict your comments to the usability, accuracy, readability, organization, and completeness of this document.

1. Please specify by page any errors you found in this manual.

2. Does the document cover the information you expected or required? Please make suggestions for improvement.

3. Is this the right type of document for your needs? Is it at the right level? What other types of documents are needed?

4. Did you have any difficulty understanding descriptions or wording? Where?

5. Please rate this document on a scale of 1 to 10 with 10 being the best rating. _____

NAME _____ DATE _____

TITLE _____

COMPANY NAME/DEPARTMENT _____

ADDRESS _____

CITY _____ STATE _____ ZIP CODE _____

Please check here if you require a written reply.

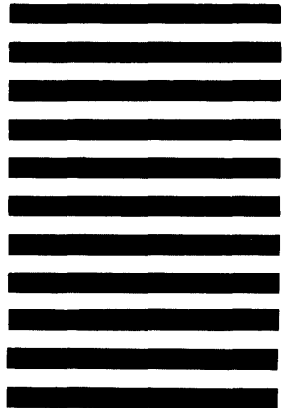
WE'D LIKE YOUR COMMENTS . . .

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

O.M.S. Technical Publications



INTEL CORPORATION, 3065 Bowers Avenue, Santa Clara, California 95051 (408) 987-8080

Printed in U.S.A.