intel

EXTENDED iRMX®II.3 OPERATING SYSTEM DOCUMENTATION

VOLUME 1 INTRODUCTION, INSTALLATION, AND OPERATING INSTRUCTIONS

Order Number: 461844-001

Intel Corporation
3065 Bowers Avenue
Santa Clara, California 95051

In locations outside the United States, obtain additional copies of Intel documentation by contacting your local Intel sales office. For your convenience, international sales office addresses are located directly before the reader reply card in the back of the manual.

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 or to keep current the information contained in this document.

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

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

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

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

Above	iLBX	iPSC	OpenNET
BITBUS	i_m	iRMX	ONCE
COMMputer	iMDDX	iSBC	Plug-A-Bubble
CREDIT	iMMX	iSBX	PROMPT
Data Pipeline	Insite	iSDM	Promware
Genius	int_el	iSSB	QUEST
i	int_elBOS	iSXM	QueX
i	Intelevision	Library Manager	Ripplemode
I^2ICE	$int_eligent$ Identifier	MCS	RMX/80
ICE	$int_eligent$ Programming	Megachassis	RUPI
iCEL	Intellec	MICROMAINFRAME	Seamless
iCS	Intellink	MULTIBUS	SLD
iDBP	iOSP	MULTICHANNEL	UPI
iDIS	iPDS	MULTIMODULE	VLSiCEL
	iPSB		

XENIX, MS-DOS, Multiplan, and Microsoft are trademarks of Microsoft Corporation. UNIX is a trademark of Bell Laboratories. Ethernet is a trademark of Xerox Corporation. Centronics is a trademark of Centronics Data Computer Corporation. Chassis Trak is a trademark of General Devices Company, Inc. VAX and VMS are trademarks of Digital Equipment Corporation. Smartmodem 1200 and Hayes are trademarks of Hayes Microcomputer Products, Inc. IBM is a registered trademark of International Business Machines. Soft-Scope is a registered trademark of Concurrent Sciences.

VOLUME PREFACE

MANUALS IN THIS VOLUME

This volume (*Introduction, Installation, and Operating Instructions*) contains the following manuals:

Introduction to the Extended iRMX® II Operating System
Extended iRMX® II Hardware and Software Installation Guide
Operator's Guide to the Extended iRMX® II Human Interface
Master Index

The *Introduction to the iRMX® II Operating System* gives a high-level description of the Extended iRMX II Operating System.

The *Extended iRMX® II Hardware and Software Installation Guide* provides instructions on how to install the Extended iRMX II Operating System on Intel microcomputers. It also provides information on how to modify both peripheral controller boards and processor boards to meet special needs.

The *Operator's Guide to the Extended iRMX® II Human Interface* provides detailed descriptions of the Human Interface and Command Line Interpreter (CLI) commands. These command are the interface between the user at a terminal and the operating system.

The *Master Index* is a guide to the entire five-volume documentation set for Extended iRMX II.3. This index provides an alphabetic list of topics followed by the title of the manual that contains the desired information.

VOLUME CONTENTS

Manuals are listed in the order they appear in the volumes. For a synopsis of each manual, refer to the *Introduction to the Extended iRMX® II Operating System.*

VOLUME 1: *Extended iRMX® II Introduction, Installation, and Operating Instructions*

Introduction to the extended iRMX® II Operating System
Extended iRMX® II Hardware and Software Installation Guide
Operator's Guide to the extended iRMX® II Human Interface
Master Index

VOLUME 2: *Extended iRMX® II Operating System User Guides*

Extended iRMX® II Nucleus User's Guide
Extended iRMX® II Basic I/O System User's Guide
Extended iRMX® II Extended I/O System User's Guide
Extended iRMX®II Human Interface User's Guide
Extended iRMX® II Application Loader User's Guide
Extended iRMX® II Universal Development Interface User's Guide
Device Drivers User's Guide

VOLUME 3: *Extended iRMX® II System Calls*

Extended iRMX® II Nucleus System Calls Reference Manual
Extended iRMX® II Basic I/O System Calls Reference Manual
Extended iRMX® II Extended I/O System Calls Reference Manual
Extended iRMX® II Application Loader System Calls Reference Manual
Extended iRMX® II Human Interface System Calls Reference Manual
Extended iRMX® II UDI System Calls Reference Manual

VOLUME 4: *Extended iRMX® II Operating System Utilities*

Extended iRMX® II Bootstrap Loader Reference Manual
Extended iRMX® II System Debugger Reference Manual
Extended iRMX® II Disk Verification Utility Reference Manual
Extended iRMX® II Programming Techniques Reference Manual
Guide to the Extended iRMX® II Interactive Configuration Utility

VOLUME 5: *Extended iRMX® II Interactive Configuration Utility Reference*

Extended iRMX® II Interactive Configuration Utility Reference Manual

RELATED PUBLICATIONS

iAPX 286 Utilities User's Guide for iRMX® II Systems

iAPX 286 System Builder User's Guide for iRMX® II Systems

iRMX Networking Software User's Guide

REV.	REVISION HISTORY	DATE
-001	Original Issue.	01/88

intel

INTRODUCTION TO THE EXTENDED iRMX®II OPERATING SYSTEM

Intel Corporation
3065 Bowers Avenue
Santa Clara, California 95051

PREFACE

INTRODUCTION

This manual is a high-level introduction to the Extended iRMX® II Operating System. It provides you with an overview of the operating system that helps you to develop your application system in less time with less expense.

READER LEVEL

This manual is written for readers who are unfamiliar with the Extended iRMX II Operating System, but who have general micro-computing experience.

MANUAL OVERVIEW

This manual is organized as follows:

Chapter 1	This chapter provides a brief introduction to the Extended iRMX II Operating System and defines terms used in later chapters.
Chapter 2	This chapter introduces some of the obstacles an operating system should try to eliminate.
Chapter 3	This chapter discusses the economic benefits of using the Extended iRMX II Operating System.
Chapter 4	This chapter discusses the features of the Extended iRMX II Operating System, and defines the vocabulary used in the other iRMX II manuals.
Chapter 5	This chapter describes a relatively simple application system.
Chapter 6	This chapter describes the other manuals associated with the Extended iRMX II Operating System.

CONVENTIONS

This manual uses the following conventions:

- Information appearing as UPPERCASE characters when shown in keyboard examples must be entered or coded exactly as shown. You may, however, mix lower and uppercase characters when entering the text.
- Fields appearing as lowercase characters within angle brackets (< >) when shown in keyboard examples indicate variable information. You must enter an appropriate value or symbol for variable fields.
- Information appearing in print indicates user input.
- The term "iRMX II" refers to the Extended iRMX II.3 Operating System.
- The term "iRMX I" refers to the iRMX I (iRMX 86) Operating System.
- All numbers unless otherwise stated are assumed to be decimal. Hexadecimal numbers include the "H" radix character (for example, 0FFH).

Intel®

CONTENTS

CHAPTER 4 (continued) **PAGE**

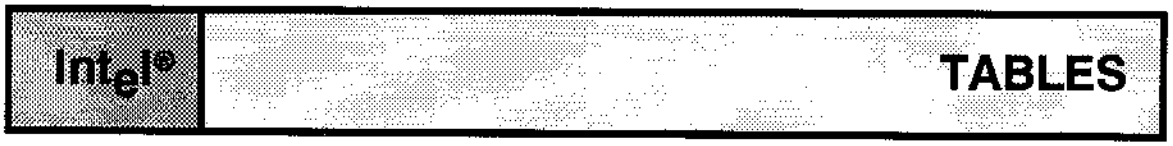


TABLE PAGE

FIGURE PAGE

CHAPTER 1
OVERVIEW OF THE iRMX® II OPERATING SYSTEM

1.1 OVERVIEW OF THE SYSTEM

The iRMX II Operating System is a software package designed for use with Intel's 80286- or 80386-based Single Board Computers and other 80286- and 80386-based microcomputers. The operating system uses the 80286 and 80386 processors in protected virtual address mode, so it takes advantage of the full memory-addressing range and protection features offered by the processor.

The iRMX II Operating System offers high-performance and real-time processing. As a real-time system, it can respond to external events called interrupts and immediately set tasks in motion to process the interrupts. Because interrupts can occur simultaneously and at seemingly random times, the operating system supports multitasking operations to handle the multiple events concurrently. In addition, the operating system allows a multiprogramming environment in which several unrelated applications can run independently.

The iRMX II Operating System is a collection of subsystems, or layers, each of which provides one or more features that can be used in your application. The reason they are called layers is that each one builds on the capabilities of the previous ones. You decide which features you need and then choose which subsystems you want. You then combine these subsystems to form a tailored operating system.

Because all of the layers except the Nucleus depend on other layers, when you include a layer (for example, the Extended I/O System), you must also include the layers it requires (the Nucleus and Basic I/O System). The following list shows the major layers of the operating system and briefly describes the purpose of each layer:

- Nucleus — The Nucleus is the heart of the iRMX II Operating System and the only required layer. All other layers of the operating system are optional.

- I/O Systems

  The I/O Systems (Basic and Extended) provide file management and the device-independent interface to input and output devices. The I/O Systems are optional components of the iRMX II Operating System, so you can exclude them if you don't need them. You can include the Basic I/O System without including the Extended I/O System, but the Extended I/O System requires the Basic I/O System.

  Device drivers are part of the Basic I/O System that provide the interface between an application and the I/O devices connected to the application. Any device drivers selected during configuration (including terminal drivers and Terminal Support Code) become part of the Basic I/O System.

- Application Loader

  The Application Loader enables your application to load programs and overlays from disk into main memory. The Application Loader is an optional part of a system, but if included, requires at least the Basic I/O System.

- Human Interface

  The Human Interface may control the application system with commands entered at a terminal. The Human Interface includes a set of commands for commonly used operations. You can also create your own commands. Like the I/O Systems, the Human Interface is an optional component and can be excluded. If the Human Interface is included, it requires all the previously described layers.

- System Debugger

  The System Debugger (SDB) extends the capabilities of the iSDM System Debug Monitor and the D-MON386 monitor by supplying "static" debugging information about the system after a "hang condition" or at any time you need to freeze and examine the system. The SDB requires only the Nucleus to run. You can include the SDB in your system during development, then remove it to reduce the size of your finished application system.

- Universal Development Interface — The Universal Development Interface (UDI) is a software interface that allows language translators and other software development tools to access the facilities of the iRMX II Operating System. The UDI is the outermost layer of any application system but may be excluded if not needed. If it is included, it requires all other system layers.

1.2 MAJOR SYSTEM FUNCTIONS

The iRMX II Operating System offers a broad range of functions. For example, the operating system can

- Simultaneously monitor and control unrelated events occurring outside the single board computer.
- Communicate with a wide variety of input, output, and mass storage devices.
- Execute on the 80286 or 80386 microprocessor in protected virtual address mode, giving it the ability to address a full 16 megabytes of memory and to provide protection features including segment length protection, stack overflow detection, invalid selector detection, and access rights protection.
- Provide a base on which to run a number of languages and other software tools.

These functions make the iRMX II Operating System an excellent foundation for your software-based products (Figure 1-1).

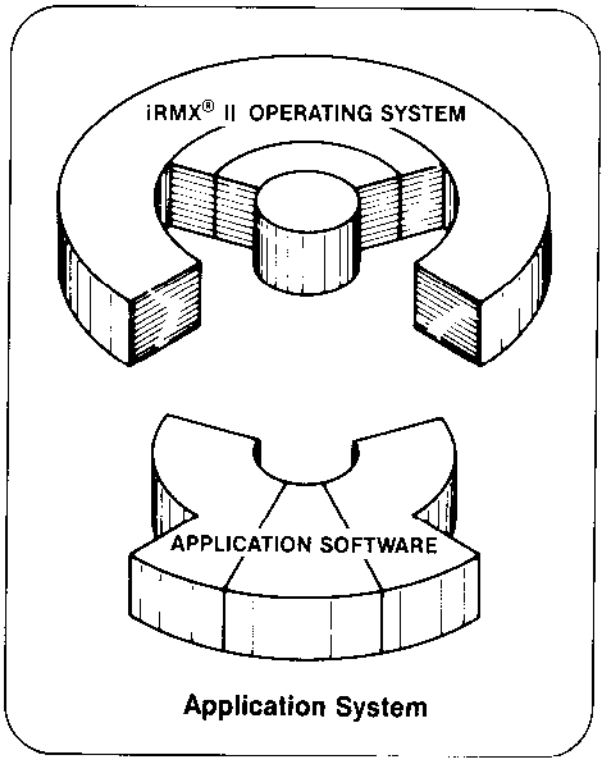


Figure 1-1. The iRMX® II Foundation for Application Systems

1.3 PROTECTED VIRTUAL ADDRESS MODE

The iRMX II Operating System runs on the 80286 or 80386 microprocessor in protected virtual address mode, enabling the operating system to take advantage of many of these microprocessor's extended features.

First, the iRMX II Operating System takes full advantage of the microprocessor's ability to address 16 megabytes of memory. This allows code and data sizes to increase over the 1 megabyte range permitted in operating systems running on 8086, 88, 186, and 188 processors or the 80286 processor running in real mode.

In addition to accessing the full range of memory, the iRMX II Operating System supports these protection features of the 80286 and 80386 processors.

- Segment-length protection that prevents segment accesses beyond their defined sizes.
- Stack-overflow detection that prevents out-of-control programs from overflowing the stack and overwriting important information.

- Invalid-selector detection that prevents programs from referring to segments of memory that haven't been defined.
- Access-rights protection that allows programs to set the read, write, or execute privileges of a segment and prevents access to those segments in other than the defined mode.

These advanced protection features, plus the ability to address the full range of memory, make the iRMX II Operating System an ideal choice for advanced applications.

1.4 iRMX® TERMINOLOGY

This manual uses the following terms frequently

Application	An application is the problem that you solve with your product.
Application Software	The application software is all the software you must add to the iRMX II Operating System to complete your application system.
Application System	An application system is the product that satisfies the requirements of the application.
Job	A job is the environment in which tasks do their work. An environment consists of tasks, the objects tasks use, a directory where tasks catalogue objects, and a memory pool. For example, a user on a multiuser system is a job.
Objects	Objects are the building blocks of the iRMX II Operating System. They are classified into the following categories: • Segments • Semaphores • Extension objects • Composite objects • Buffer pools • Mailboxes • Regions • Jobs • Tasks
Object-oriented Architecture	Object-oriented architecture is a system design marked by a relationship between data and a token for the data. This relationship gives form to the data and relieves the programmer from having to provide this structure.

Protected Virtual Address Mode	Protected virtual address mode supports 16 megabytes of physical memory in RAM or ROM and 1 gigabyte of virtual memory per user. It provides an on-chip memory management facility that translates virtual addresses to physical memory addresses. It also protects the operating system from unauthorized modification by application programs and isolates each user from other users. Protected Virtual Address Mode is supported in both 80286- and 80386-based systems.
Real Address Mode	Real address mode is the method of 80286 execution that supports 1 megabyte of physical memory in RAM or ROM. The iRMX II Operating System does not run in real address mode except for a short time when the system boots up. However, 8086 or 8088 programs that do run in real address mode can execute in an iRMX II system with some slight modification. Refer to the *Extended iRMX II Programming Techniques Reference Manual* for a description of converting iRMX I applications to iRMX II applications.
Segments	Segments are contiguous pieces of memory existing within the environment of the job in which they were created. They form the fundamental piece of system memory for such uses as task stacks, data storage, and system buffers.
Tasks	Tasks are the active objects that do the work of the system. Tasks are written as parameterless procedures.
User	The user is the individual or organization who uses your application system.

CHAPTER 2
CONSIDERATIONS RELATING TO REAL-TIME SOFTWARE

2.1 INTRODUCTION

The problems encountered in real-time programming differ from those found in other types of programming. This chapter briefly introduces some of the problems that face designers of real-time systems. Note that this chapter only poses questions--it provides no answers. You can find the answers in the discussion of iRMX II features in Chapter 4.

2.2 EVENT DETECTION

Real-time application systems monitor events in the real world. These events occur asynchronously, that is, at seemingly random intervals. When an event occurs, the system could be in the midst of processing information associated with a previous event. So the system must be able to detect and record the occurrence of the second event without affecting the processing associated with the previous event.

2.3 SCHEDULING OF PROCESSING

Assuming that the system can detect and record the occurrence of an event, it still must decide in what order to process recorded events. For example, when the system is processing a relatively unimportant event and a critical event occurs, it must be able to postpone processing of the less significant event until the more important one has been processed. Then it must resume where it left off with the less significant event.

2.4 ERROR PROCESSING

Suppose that during the processing of real-time events, an error is detected. How can the error be corrected, or how can its impact be limited, without adversely affecting the system? The whole system, for instance, should not be shut down merely because an error is detected; the system should be able to recover from these errors and continue processing.

2.5 DEVICE INDEPENDENCE

Many real-time applications use one or more input or output devices. Occasionally the devices associated with an application system must be changed. By allowing devices to be changed without requiring recompilation, the operating system can save time and effort.

2.6 MASS STORAGE FILE ALLOCATION TRADEOFFS

In any real-time system, file allocation performance is an important consideration. One factor that relates directly to mass storage file allocation performance is the size of each contiguous piece of data written to and read from a file (the file's "granularity"). In some applications, large granularity results in much faster retrieval. In other applications, large granularity does not improve performance, but does waste space on the storage device. The operating system must contend with the tradeoff between performance and optimal use of space on the device.

2.7 FEATURE SELECTION

An operating system should be flexible enough to let you select required features and eliminate unneeded features. Because operating systems are complex, the process used to select features should be efficient, easy to use, and easy to understand.

2.8 MULTIPLE APPLICATIONS

Sometimes you may need to run more than one application on the same computer. Several applications might need to share some resources, such as hardware and perhaps some files, while reserving other resources for themselves.

2.9 MEMORY REQUIREMENTS

The memory requirements of some applications change according to the events that occur in the real world. If a system can share memory between applications, then the total amount of memory required for the system might be less than the sum of the maximum amounts required by each application.

2.10 FILES AND MULTIPLE USERS

Some applications, such as data entry and database management systems, support more than one user at a time. In such systems, three major problems must be dealt with:

- File naming--users must be able to name files without concern for duplicate names. If they cannot, each user may be forced to create unique names.

- Selective file sharing--multi-user systems often must be able to share and protect files. For instance, in a data entry system, one operator may be entering data while another simultaneously verifies the entered data (file sharing). Or perhaps the file contains confidential information. Once verified, the file must be protected against unauthorized reading and writing (file protecting).
- Responding simultaneously to more than one terminal--the system must respond quickly to each terminal and must be able to keep track of tasks and other resources associated with a particular terminal.

2.11 THE HUMAN ELEMENT

Applications must be controlled by people. Systems often contain critical processes that operators must control with a minimum chance of error. An application system should provide a set of interactive commands and messages that are easy to use and understand.

2.12 APPLICATION DEVELOPMENT

Frequently the hardware on which an application system is installed (called the target system) includes mass storage devices and file structures. The operating system running on the target system should allow application system development using existing hardware. This process, termed on-target development, enables you to use language processors (such as assemblers, compilers, and run-time support systems), linking utilities, editors, and file maintenance utilities. You should be able to install such development tools on the operating system quickly and easily.

2.13 DEBUGGING

Object-oriented application systems require debugging tools that are sensitive to the objects that make up the operating system. Tools such as these allow engineers to concentrate on the logic errors or "bugs" in their code instead of being distracted by the details of operating-system implementation code. If the system crashes (becomes inoperative), it might be necessary to "freeze" the system and examine its state. This type of debugging is called "static" debugging.

2.14 HARDWARE BUS ARCHITECTURE

To handle the needs of your hardware, a real-time software package should support industry standard buses, or a custom bus, without sacrificing performance or usability.

CHAPTER 3
BENEFITS OF THE iRMX® II OPERATING SYSTEM

3.1 INTRODUCTION

By serving as a foundation for your application software (Figure 3-1), the iRMX II Operating System can help you develop your application system quickly using the latest technology, minimize your development costs, and minimize your costs after development.

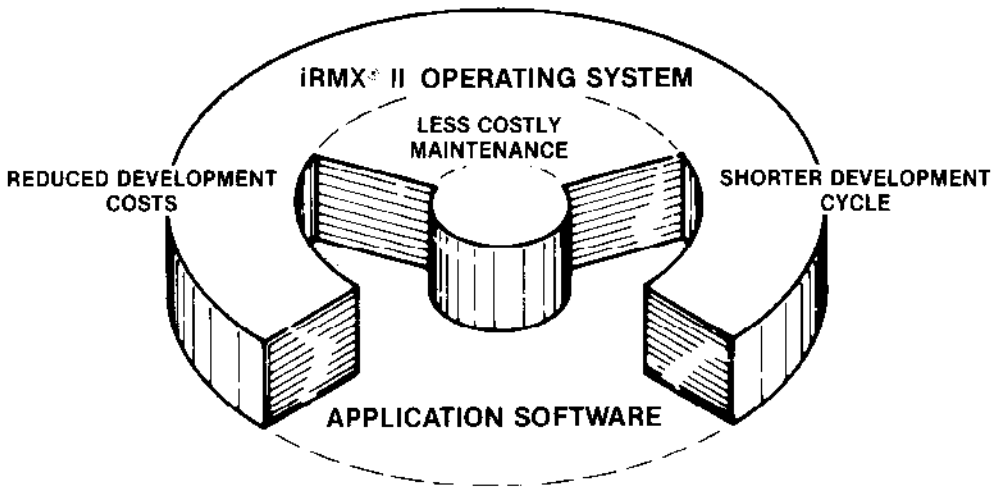


Figure 3-1. The iRMX® II System Provides Economic Benefits

3.2 DEVELOPMENT TIME

The iRMX II Operating System helps you develop real-time application systems quickly. As the base for your application software, the operating system provides services required by many real-time applications. Since these services are already supplied, application engineers spend no time writing software to manage real-time functions such as multitasking and dynamic memory allocation and can instead concentrate on the software that relates specifically to the application. This concentration of effort greatly reduces the time needed to develop your application system.

3.3 COST OF IMPLEMENTATION

The iRMX II Operating System helps reduce the cost of implementation in the following ways:

- By supplying the general services required by many real-time applications, the operating system reduces your manpower requirements.
- Industry-standard languages are available for use with the operating system.
- The features of the operating system simplify the process of development. Chapter 4 discusses these features, which include object-oriented architecture and device independence.
- Support for the latest 86 family microprocessors is available now, resulting in immediate improvements in speed and performance.

3.4 COSTS AFTER DEVELOPMENT

After your application system is developed, your major expense is maintenance--the process of correcting logic errors, making changes, and adding features. The iRMX II Operating System helps minimize these costs in the following ways:

- A number of features, such as multitasking and multiprogramming, smooth the process of system design, reducing the probability of major design errors. Chapter 4 discusses these features.
- When errors do reveal the presence of bugs in your application software, you need tools to help find the errors. The operating system provides these tools, which include error handlers, a static system debugger, and an interactive debugger. Chapter 4 also discusses these features.
- The modularity provided by multiple jobs and tasks enables you to make changes and additions without severely affecting the system's overall design. This feature is a direct result of an object-oriented system. Chapter 4 discusses object-oriented architecture.

CHAPTER 4
FEATURES OF THE iRMX® II OPERATING SYSTEM

4.1 INTRODUCTION

This chapter discusses the major features of the iRMX II Operating System. Figure 4-1 presents a conceptual view of these features.

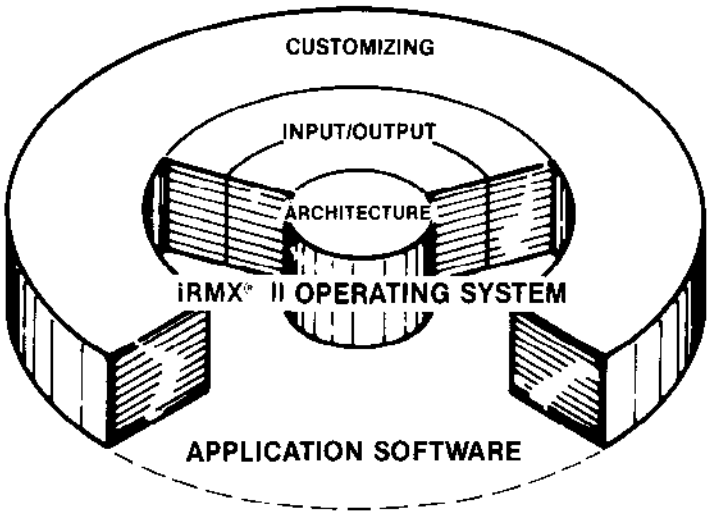


Figure 4-1. Features of the iRMX® II Operating System

FEATURES OF THE iRMX® II OPERATING SYSTEM

This chapter describes the following features:

ARCHITECTURAL FEATURES

- Object-Oriented Architecture
- Multitasking
- Interrupt Processing
- Pre-emptive Priority-Based Scheduling
- Round-Robin Scheduling
- Multiprogramming
- Intertask Coordination
- Bus Architecture Support
- Extendibility

INPUT/OUTPUT FEATURES

- Choice of I/O Systems
- Device-Independent Input and Output
- Hierarchical Naming of Mass Storage Files
- File Access Control
- Control over File Fragmentation
- Selection of Device Drivers
- Remote Files using iRMX-NET®
- Terminal Support Code

CUSTOMIZING FEATURES

- Custom Interactive Commands
- Applications Loading
- Terminal Support
- Run-Time Binding
- Error Handling
- Dynamic Memory Allocation
- Bootstrap Loading

TOOLS

- System Debugger
- Soft-Scope® 286
- Start-up Systems
- Interactive Configuration Utility (ICU)
- File Maintenance Programs

ON-TARGET PROGRAM DEVELOPMENT

4.2 ARCHITECTURAL FEATURES

When Intel software engineers designed the iRMX II Operating System, they specified the basic processes and data structures of the system, including such characteristics as the partitioning of programs into "tasks," task scheduling, and task communication. These characteristics are referred to as the "architecture" of the system. The important architectural features of the operating system are described here.

4.2.1 Object-Oriented Architecture

An object-oriented solution uses a collection of smaller solvable tasks or objects to collectively solve a much larger more complex problem. For example, suppose you have a problem that needs solving. On the whole, the problem is very large and complex. To solve it with one solution of entwined hardware and software would be a very difficult task. Because of the complexity of the problem, the length of design would be long and the chance for system problems great. Furthermore, once the problem was solved and the design of the system complete, any change to the system or the requirements of the problem would cause a massive rippling effect of changes throughout the system. For instance, something as common as changing a hardware device could cause coding changes throughout the entire system.

An object-oriented solution, however, breaks the large complex problem up into smaller more manageable tasks or objects. Each task can then be designed with minimum dependence upon other tasks within the system. Taken all together, the effort of the many smaller tasks provides the solution for the original complex problem.

The iRMX II Operating System uses an object-oriented architecture designed to facilitate object-oriented solutions. The operating system is object-oriented because it supports small individual tasks by providing system calls to manage such things as task communication, regions of memory, and exception handlers.

You can view the iRMX object-oriented architecture as a collection of defined building blocks manipulated by users. The building blocks of the iRMX II Operating System are called objects and are of several types. Some object types are tasks, jobs, mailboxes, semaphores, segments, buffer pools, and connections. The characteristics of the building blocks are easy to learn and subsequently use to build a solution to your problem. The reason for their ease of use is because they are well-defined and consistent. Each object type (such as a mailbox), for example, has a specific set of attributes. Once you become familiar with the attributes of a mailbox, you are familiar with all mailboxes. There are no special cases. Also, each type of iRMX II object has an associated set of system calls. These calls cannot be used to manipulate objects of another type without causing an error. Thus, you are guided into compliance with the rules of the operating system.

The objects in an iRMX system are acted on by system calls. In other words, your application software uses system calls to manipulate the tasks or objects in your application system. For instance, the CREATE MAILBOX and DELETE MAILBOX system calls do precisely what their names suggest. They create and delete communication vehicles called mailboxes that are used between two tasks.

Figure 4-2 summarizes how a problem that requires several types of input to be processed into several types of output can be solved in an object-oriented operating system. Each input task is a complete solution in itself to the problem of receiving and processing different types of data. After each task processes its data, the data exits the task using a similar format. System calls are responsible for moving data from input tasks to a waiting mailbox (object). From here, a manager task gathers the like messages and processes them. Depending upon where the data is to be sent, the manager task sends the similar data messages to the appropriate output task. Finally, the individual output tasks receive and process their data.

It is important to note that because the data can be sent in a universal format, the input and output tasks are independent of the manager task. This fact eliminates dependency problems between tasks. Also, the object-oriented architecture of the operating system is designed to easily create mailboxes, tasks, and other objects needed to create the environment necessary to provide a solution.

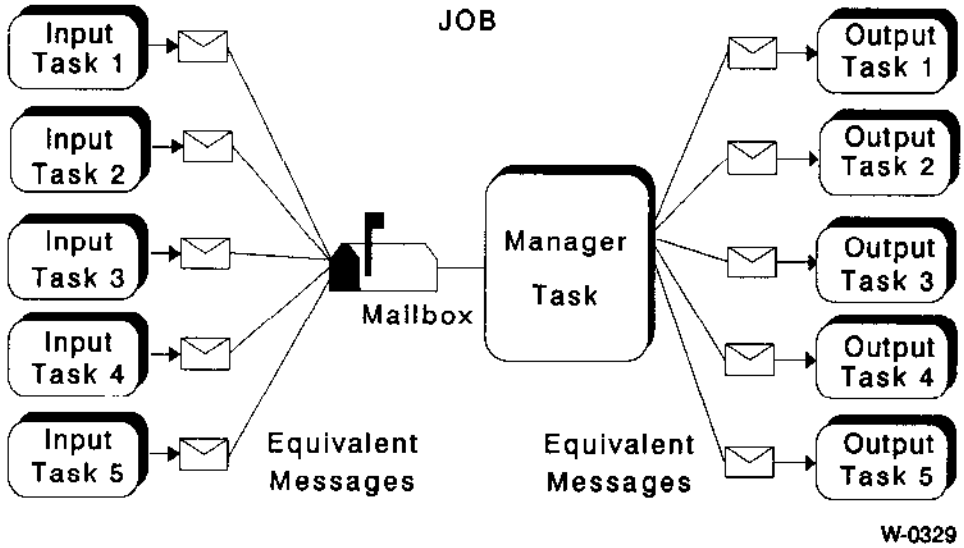


Figure 4-2. Object-Oriented Solution

4.2.2 Multitasking

A real-time application system can process numerous events occurring at seemingly random times. These events are asynchronous because they can occur at any time, and they are potentially concurrent because one event might occur while another is being processed.

Any single program that attempts to process multiple, concurrent, asynchronous events is bound to be complex. The program must perform these functions:

- process events
- remember which events have occurred and the order in which they occurred
- remember which events have occurred but have not been processed

As the system monitors more events, the complexity of the program increases.

Multitasking is a technique that unwinds this confusion. Rather than writing a single program to process a number of events, you can write individual programs, each of which processes a single event. Each of these individual programs forms an iRMX II task, one of the types of objects mentioned earlier. In many cases, this technique eliminates the need to monitor the order in which events occur.

Multitasking simplifies the process of building an application system, enabling you to build your system faster and at less expense. Also, because of the one-to-one relationship between events and tasks, your system's code is less complex and easier to maintain.

4.2.3 Interrupt Processing

The iRMX II Operating System is an interrupt processor. When an interrupt occurs, the operating system schedules a task to process the interrupt. This method of event detection improves the performance of your application system.

Computer systems detect and control events in the real world using two processing schemes: polling and interrupt. In polling, the software periodically checks to see if certain events have occurred. For example, imagine a class of students and a teacher. If, rather than spotting raised hands, the instructor specifically asks each student in the class if the student has any questions, then the instructor is polling the students.

Polling has major shortcomings: two of which are a significant amount of the processor's time is spent testing to see if events have occurred, and immediate attention cannot be given to an interrupt that needs servicing immediately.

Interrupt processing is the processing scheme that the iRMX II Operating System uses. When an event occurs, the processor is literally interrupted. Rather than executing the next sequential instruction, the processor begins to execute a task associated specifically with the detected event.

The example of a classroom can illustrate interrupt processing. If a student has a question, she raises her hand and speaks the instructor's name. The instructor, interpreting this as an interrupt, finishes his sentence and deals immediately with the student's question. Once the instructor has answered the student's question, he returns to what he was doing before he was interrupted.

Interrupt processing of external events provides your application system with the following benefits:

- More Efficiency — Interrupt processing enables your system to spend all of its time running the tasks that process events, rather than executing a polling loop to see if events have occurred.

- More Flexibility — Because of the direct correlation between interrupts and tasks, your system can easily be modified to process different events. All you need to do is write the tasks to process the new interrupts.
- Economic Benefits — Because interrupt processing enables your system to respond to events using modularly coded tasks, your system's code is more structured and easier to understand than monolithic code. Modular code is less costly to develop and maintain and can be developed more quickly than monolithic code.
- Better Performance — Faster response to external events results in better performance.

4.2.4 Scheduling Algorithms

The iRMX II Operating System uses a combination of two scheduling algorithms to determine which task runs at any instant. It uses pre-emptive, priority-based scheduling to determine when tasks of unequal priority will run. It also includes a round-robin scheduling feature that grants tasks of equal priority equal access to the processor.

4.2.4.1 Pre-emptive, Priority-Based Scheduling

When the priorities of tasks are different, the iRMX II Operating System uses pre-emptive priority-based scheduling to decide which task runs at any instant. This technique ensures that if a more important task becomes ready while a less important task is running, the more important task begins execution immediately.

In multitasking systems, there are two common techniques for deciding which task is to be run at any given moment. Time slicing, where tasks are run in rotation for a period, or "slice" of time, is the technique used in time-sharing systems (the iRMX II Operating System uses a variation of this technique to decide among equal-priority tasks). The second technique, priority-based scheduling, uses assigned priorities to decide in which order tasks are run.

Within priority-based scheduling are two approaches: Nonpre-emptive scheduling and pre-emptive scheduling. Nonpre-emptive scheduling allows a task to run until it relinquishes the processor. Even if a higher-priority task becomes ready for execution, the original task continues to run until completion. With pre-emptive scheduling, the system always executes the highest-priority task that is ready to run. In other words, if the running task or an interrupt causes a higher-priority task to become ready, the operating system switches the processor to the higher-priority task.

Pre-emptive, priority-based scheduling goes hand-in-hand with the interrupt processing discussed earlier. Task priorities can be tied to the relative importance of the events that they process. This enables the processing of more important events to pre-empt the processing of less important events without abandoning the less important events.

4.2.4.2 Round-Robin Scheduling

In addition to pre-emptive, priority-based scheduling, the iRMX II Operating System includes round-robin scheduling to ensure that equal-priority tasks all get a chance to run. With round-robin scheduling, each task is allocated a time quota. When the time quota expires, the task is preempted and the next task of the same priority is allowed to run. This technique allows equal-priority tasks to take turns running.

Of course, higher priority tasks can still preempt any running task, regardless of the amount of time left in its quota. However, without this feature, when the higher-priority task finishes, the first task regains control and can continue running indefinitely until another higher-priority task preempts it. This blocks out all other tasks of the same priority.

Round-robin scheduling is especially useful for multi-user systems, in which several users might have tasks running at the same priority. Round-robin scheduling prevents one user's processor-intensive task (such as a program compilation) from stopping other users' work.

Round-robin scheduling is not desirable for high-priority tasks, such as interrupt tasks, where immediate response is crucial. Therefore, the round-robin feature is normally configured to take effect only for tasks whose priorities are below a specified threshold. The length of the time quota can also be adjusted.

4.2.5 Multiprogramming

Multiprogramming is a technique used to run several applications on a single application system, thus using the system hardware more fully. To take full advantage of multiprogramming, you must provide each application with a separate environment; that is, separate memory, files and objects. This isolation prevents independently developed applications from causing problems for each other.

For instance, suppose that two unrelated applications share a temporary file on a disk. If the first application writes information to the file and the second application writes over the file, the first application has problems. The only way to avoid this kind of problem with shared files is to create some form of mutual exclusion. But if the two applications must interact even to the point of excluding each other, they cannot be developed independently. The two engineers creating the applications must coordinate with each other and spend valuable time that could be used within, rather than between, applications. The only alternative would be to avoid sharing the file.

The iRMX II Operating System provides a type of object that can be used to obtain this kind of isolation. This object, called a job, has these characteristics:

- Unlike tasks, jobs are passive. They cannot invoke system calls.
- Each job includes a group of tasks and the resources they need.
- Jobs serve as useful boundaries for dynamically allocating memory. When two tasks of one job request memory, they share the memory associated with their job. Two tasks in different jobs do not directly compete for memory.
- One or more jobs make up an application.
- Each job serves as an error boundary. When the application detects an error, or when the operator decides to abort an application, a job is a convenient object to delete.

Multiprogramming provides your application system with two benefits:

- It increases the amount of work your system can do. By using your hardware more fully, your system can run several applications rather than one. Thus, more processing is squeezed out of your hardware investment.
- Because of the correspondence between jobs and applications, new jobs can be added to your system (or old jobs removed) without affecting other jobs. This makes your system much easier and faster to modify.

4.2.6 Inter-Task Coordination

The iRMX II Operating System provides simple techniques for tasks to coordinate with one another. These techniques allow tasks in a multitasking system to mutually exclude, synchronize, and communicate with each other.

As we have already seen, multitasking is a technique used to simplify the designing of real-time application systems that monitor multiple, concurrent, asynchronous events. Multitasking enables engineers to focus their attention on the processing of a single event rather than having to contend with numerous other events occurring in an unpredictable order.

However, the processing of several events may be related. For instance, the task processing one event may need to know how many times another event has occurred since the first event last occurred. This kind of processing requires that tasks be able to coordinate with each other. The iRMX II Operating System provides for this coordination.

Tasks interact with each other in three ways: exchanging information, mutually excluding each other, and synchronizing with each other.

4.2.6.1 Exchanging Information

Tasks exchange information for two purposes. One purpose is to pass data from one task to another. For instance, suppose that one task accumulates keystrokes from a terminal until it receives a carriage return. It then passes the entire line of text to another task that is responsible for decoding commands.

The second reason for passing data is to draw attention to a specific object in the application system. In effect, one task says to another, "I am talking about that object."

The iRMX II Operating System facilitates intertask communication by supplying objects called "mailboxes" along with system calls that manipulate mailboxes. The system calls associated with mailboxes are CREATE MAILBOX, DELETE MAILBOX, SEND DATA, RECEIVE DATA, SEND MESSAGE, and RECEIVE MESSAGE. Tasks use the first two system calls to build and remove a particular mailbox. They use the remaining calls to communicate with each other (SEND/RECEIVE DATA and SEND/RECEIVE MESSAGE are simply two different ways of communicating via a mailbox).

Let's see how tasks can use a mailbox for communicating and for sending information. If Task A wants Task B to become aware of a particular object, Task A uses the SEND MESSAGE system call to mail that object to the mailbox. Task B uses the RECEIVE MESSAGE system call to get the object from the mailbox. (Note that this example is somewhat simplified to serve as an introduction. For detailed information, see the *Extended iRMX II Nucleus User's Guide*.)

Tasks can also use mailboxes to send information to each other. If Task A wants to send information to Task B, it can use the SEND DATA system call to mail that information to a mailbox. Task B uses the RECEIVE DATA system call to retrieve the information from the mailbox.

Why don't tasks talk directly to each other, rather than through mailboxes? Tasks are asynchronous--they run in an unpredictable order.

If two tasks want to communicate with each other, they need a place to store messages and to wait for messages. If the receiver uses the RECEIVE MESSAGE system call before the message has been sent, the receiver can wait at the mailbox until a message arrives. Similarly, if the sender uses the SEND MESSAGE system call before the receiver is ready to receive, the message is held at the mailbox until a task requests a message from the mailbox. In other words, mailboxes allow tasks to communicate with each other even though tasks are asynchronous.

4.2.6.2 Mutual Exclusion

Occasionally, when tasks are running concurrently, the following kind of situation arises:

1. Task A is reading information from a memory segment.
2. An interrupt occurs and Task B, which has higher priority than Task A, pre-empts Task A.
3. Task B modifies the contents of the memory segment that Task A was in the midst of reading.
4. Task B finishes processing its event and surrenders the processor.
5. Task A resumes reading the memory segment.

The problem is that Task A might now have invalid information. For instance, suppose the application is air traffic control. Task A is responsible for detecting potential collisions, and Task B is responsible for updating the Plane Location Table with the new X- and Y-coordinates of each plane's location. Unless Task A can gain exclusive use of the Plane Location Table, Task B can make Task A fail to spot a collision.

Here's how it could happen. Task A reads the X-coordinate of the plane's location and is pre-empted by Task B. Task B updates the entry that Task A was reading, changing both the X- and Y-coordinates of the plane's location. Task B finishes its function and surrenders the processor. Task A resumes execution and reads the new Y-coordinate of the plane's location. Since Task B changed the Plane Location Table while Task A was reading it, Task A now thinks the plane is at old X and new Y.

This problem can be avoided by mutual exclusion. If Task A can prevent Task B from modifying the table until after Task A has finished using it, Task A can be assured of valid information. Somehow, Task A must obtain exclusive use of the table.

The iRMX II Operating System provides two types of objects that can be used to provide mutual exclusion: the semaphore and the region. A semaphore is an integer counter that tasks can manipulate using four system calls: CREATE SEMAPHORE, DELETE SEMAPHORE, SEND UNITS, and RECEIVE UNITS. The creation and deletion system calls are used to build and remove semaphores. The send and receive system calls can be used to achieve mutual exclusion.

Regions allow tasks to share data. Mutual exclusion is achieved because only one task may access a region at a time. The use of regions should be restricted to programmers who have a firm understanding of the iRMX II Operating System. For more information on regions, see the *Extended iRMX II Nucleus User's Guide*.

Before discussing how semaphores can provide exclusion, we must examine their properties. As mentioned above, a semaphore is a counter. It can take on only nonnegative integer values. Tasks can modify a semaphore's value by using the SEND UNITS or RECEIVE UNITS system calls. When a task sends n units (n must be zero or greater) to a semaphore, the value of the counter is increased by n. When a task uses the RECEIVE UNITS system call to request x units (x must be zero or greater) from a semaphore, one of the following two things happens:

- If the semaphore's counter is greater than or equal to x, the operating system reduces the counter by x and continues to execute the task.
- Otherwise, the operating system begins running the task having the next highest priority, and the requesting task waits at the semaphore until the counter reaches x or greater.

How can tasks use a semaphore to achieve mutual exclusion? Create a semaphore with an initial value of 1. Before any task uses the shared resource, it must receive one unit from the semaphore. As soon as a task finishes using the resource, it must send one unit to the semaphore. This technique ensures that at any given moment, no more than one task can use the resource, and any other tasks that want to use it await their turn at the semaphore.

Semaphores allow mutual exclusion; they don't enforce it. All tasks (there can be more than two) sharing the resource must receive one unit from the semaphore before using the resource. If one task fails to do this, mutual exclusion is not achieved. Also, each task must send a unit to the semaphore when the resource is no longer needed. Failure to do this can permanently lock all tasks out of the resource.

4.2.6.3 Synchronization

As mentioned earlier, tasks are asynchronous. Nonetheless, occasionally a task must know that a certain event has occurred before the task starts running. For instance, suppose that a particular application system requires that Task A cannot run until after Task B has run. This kind of requirement calls for synchronizing Task A with Task B.

Your application system can achieve synchronization by using semaphores. Before executing either Task A or Task B, create a semaphore with an initial value of zero. Then have Task A issue RECEIVE UNITS requesting one unit from the semaphore. Task A is forced to wait at the semaphore until Task B sends a unit. This achieves the desired synchronization.

Semaphores and mailboxes can accommodate a wide variety of situations. The only limit on the number of mailboxes and semaphores is the maximum number of objects allowed in the system (2000H).

4.2.7 Bus Architecture Support

To handle the needs of your hardware, the iRMX II Operating System supports two industry-standard bus architectures: MULTIBUS® I and MULTIBUS II.

Some benefits and features of the MULTIBUS I architecture as follows:

- Use of a 16-bit wide data word.
- A wide installation base exists. MULTIBUS I architecture has existed for some time and is very common in the industry.
- The bus offers intelligent board-to-board communications via hardware signals and similar signal processing techniques between boards.

Some benefits and features of the MULTIBUS II architecture are as follows:

- Use of a 32-bit wide data word.
- Enhanced board-to-board communication through additional internal buses and a well-defined data transfer protocol. The internal buses allow for a virtual interrupt processing scheme that lets any board communicate with any other, regardless of hardware limitations, such as eight physical interrupt lines used in MULTIBUS I systems.
- Efficient bus use through a "data packet" transfer scheme. Transferring data by packets (in smaller portions) eliminates a slower device on the bus from stealing all the bus time. With this transfer method, the bus is no longer limited in speed to the slowest device using it.

4.2.8 Extendibility

Something is extendible if you can add to it. Since the iRMX II Operating System is extendible, system programming engineers can build their own types of objects and the system calls to manipulate those objects. These custom features become a part of the operating system. From the programmer's point of view, there is no way to distinguish custom objects from those supplied by Intel.

The advantage of extendibility is that you can add your own features to the iRMX II Operating System and obtain the same benefits as supplied by its object-oriented architecture. These benefits include the ability to send custom-made objects to mailboxes and put them in object directories. Also, application engineers can more quickly become familiar with those custom features. This familiarization shrinks development time and costs, enabling you to bring your application system to market sooner.

4.3 INPUT/OUTPUT FEATURES

Input and output operations are a large part of most applications, so the operating system offers a collection of I/O features to speed development of application systems, and to make the I/O of those systems efficient.

4.3.1 Choice of I/O Systems

To meet the I/O needs of a wide variety of applications, the iRMX II Operating System provides two I/O systems: the Basic I/O System (BIOS) and the Extended I/O System (EIOS). You can use only the Basic I/O System, or you can combine the two. You cannot use only the Extended I/O System.

4.3.1.1 Basic I/O System (BIOS)

For some applications, the performance or flexibility of the system is more critical than the time necessary to produce the system. For these applications, the iRMX II Operating System provides the Basic I/O System.

The BIOS is the more flexible of the two I/O systems. It provides very powerful capabilities, and it makes few assumptions about the requirements of your application. The following features illustrate the flexibility of the BIOS:

- Enables Design of Buffering Algorithm — Rather than automatically providing a buffering algorithm, the BIOS enables you to design and implement your own buffering technique. Using the BIOS, you control the synchronization between I/O and processing.
- Appropriate for Random I/O Operations — Perhaps the I/O in your application is random access. This means that rather than reading or writing data in sequential blocks, the application accesses data in blocks that are not adjacent to each other. The BIOS is more appropriate for these operations because of the explicit control the programmer has over I/O operations.
- Gives Task Control of I/O Details — The system calls of the BIOS often have many parameters. Using these parameters, your tasks can closely tailor the behavior of each system call to match the performance requirements of your application system.

The BIOS emphasizes flexibility rather than ease of use. It provides I/O features useful in time-critical or memory-critical applications, and enables the performance of a system to be optimized.

4.3.1.2 Extended I/O System (EIOS)

The Extended I/O System is designed to be easy to use and efficient for sequential I/O. Here are some of the important features of the EIOS:

- Automatic Buffering of I/O Operations

  If you want to use multiple-buffered I/O but do not want to be burdened with writing complex code to check and switch buffers, you can use EIOS system calls. When the application program issues a system call to perform an I/O operation, the operating system performs the input or output operation and then returns control to the user program after the data transfer is complete. Before returning control to the user program, the operating system starts reading or writing the next block.

  For example, if the application is reading a file from disk, the following sequence will occur:

  1. When the application program opens a file using an EIOS system call, the operating system starts reading the first block of the file ("initiates" the input).

  2. The operating system returns control to the application program.

  3. Later, the program requests an EIOS read. The operating system has already started reading this data. When the input is complete, the operating system initiates a read of the next block of the file (called "reading ahead") and returns control to the calling program.

  Thus, whenever the user requests an EIOS read, the data is either immediately available or is in the process of being read.

  The equivalent output process is performed by "writing behind." When an application program requests an EIOS write, the operating system copies the data to a buffer maintained by the EIOS and returns to the calling program. Whenever this buffer is filled, the system initiates an output operation.

- Efficient Sequential I/O Operations — Another characteristic of the EIOS is that when it does a "read ahead" operation, the operating system assumes that a series of sequential reads are to be performed. For example, the operating system will read data from disk address 23, then from disk address 24, and so on. So, when your I/O is mostly sequential (for example, if you were examining consecutive records of a file), EIOS system calls are particularly efficient. You can also perform random access of a file with the EIOS by preceding operations with a seek call specifying the offset into the file. This method, however, is less efficient than using the BIOS.

- Free of Tedious Details — The system calls of the EIOS have relatively few parameters and are easy to code. In most cases, a single EIOS call serves the purpose of several BIOS calls. This correlation simplifies your application system, which reduces development time and costs.

The iRMX II Operating System enables you to select the features you want. The BIOS gives maximum control of I/O operations for applications requiring finely tuned performance, especially while doing random access I/O. The EIOS is easy to use and saves in cost and development time, especially in applications that use sequential I/O.

Finally, you can use both I/O systems when your application uses I/O for several purposes, some that can best be done by the Basic I/O System and some by the Extended I/O System.

4.3.2 Device-Independent Input and Output

A system provides device-independent I/O if it has one set of system calls for communicating with all I/O devices. The alternative to device independence is to provide different calls for each type of device. Let's first examine the alternative and then move on to device independence.

Consider an operating system that does not provide device independence. The system calls controlling I/O are explicitly related to the I/O devices being used. For instance, the system call for writing to the line printer might be PRINT, while the system call for writing to the terminal might be TYPE. Once you have written a procedure in such a system, the procedure is locked into a particular combination of devices. The only way you can reroute input or output is to regenerated the operating system and edit and recompile the application source code.

Now consider a device-independent operating system: the iRMX II Operating System. Because it supports device-independent I/O, the system calls are not device-dependent. The READ system call is always used for input, and the WRITE system call is always used for output. The device is specified by a parameter within the system call. Consequently, by using a variable as the parameter that selects the device, you create I/O procedures that are completely independent of the devices they use.

Device independence makes your application system very flexible. If you write a procedure to log events on a line printer, you can use the same procedure to log events on a terminal or on a disk. You need not regenerate your operating system, but only need to edit and recompile the application source code.

4.3.3 Hierarchical Naming of Mass Storage Files

Hierarchical naming is one of three common techniques used to name files on mass storage devices such as disks and bubble memories. The other two techniques are called simple naming and directory naming. The advantages of hierarchical naming become clear when comparing them to the other two naming methods. First, we'll look at simple naming.

Simple naming enables you to provide files with a descriptive name. For instance, you might decide to name files ACCOUNTS PAYABLE, ACCOUNTS RECEIVABLE, TRANSACTIONS, and INVENTORY. These names are certainly descriptive, but what happens when a different application running in the same system also decides to use one of these names? A conflict occurs. The use of a more powerful naming technique avoids this conflict.

Directory naming allows different applications (or even different users) to use the same file name. Each application (or user) is given one special-purpose file called a directory. This directory contains only file names; it does not contain data. Figures 4-3 and 4-4 provide examples of directories. When application software refers to a specific file, it first names the directory and then names the file. For instance, in Figure 4-3, the TRANSACTIONS file associated with Engineering would be designated ENGINEERING/TRANSACTIONS. The comparable file for Marketing, in Figure 4-4, would be designated MARKETING/TRANSACTIONS.

The advantage of directory naming over simple naming is directory naming allows file names to reflect relationships between files. In Figure 4-3, files related to Engineering are in the directory called ENGINEERING. This grouping of related files is not supported by simple naming.

What about situations where more than one level of directory is required? This is illustrated in Figure 4-5, which differs from Figure 4-4 only in that a second level of grouping has been included.

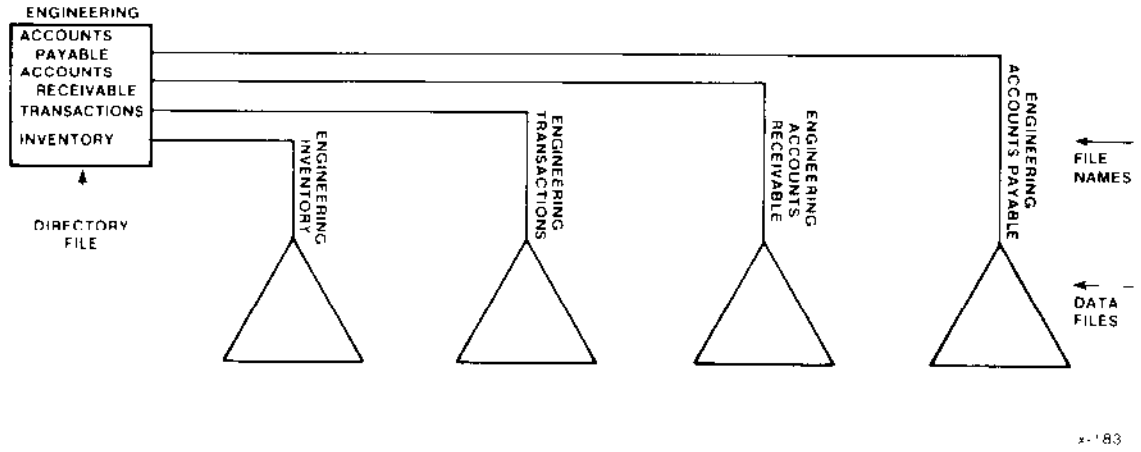


Figure 4-3. An Engineering Directory

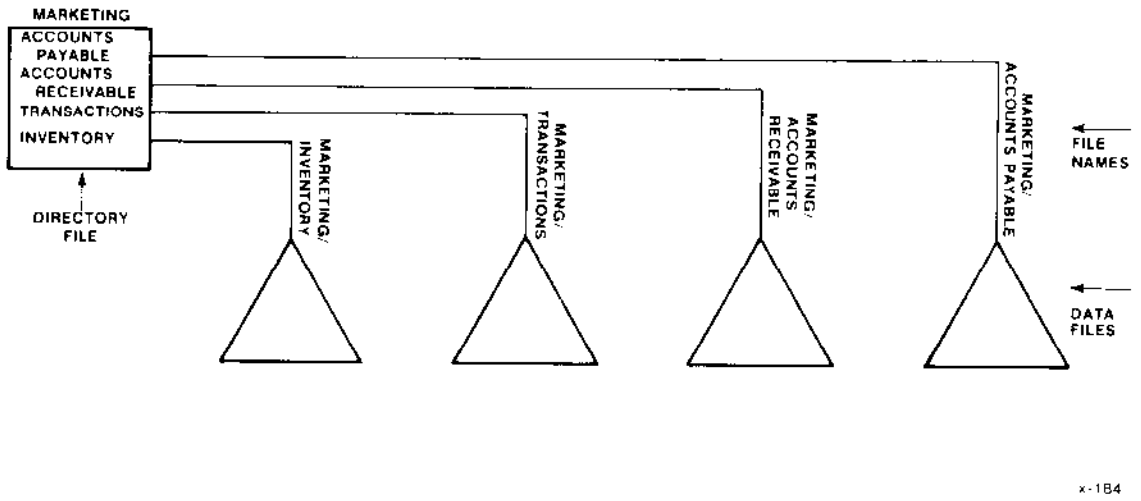


Figure 4-4. A Marketing Directory

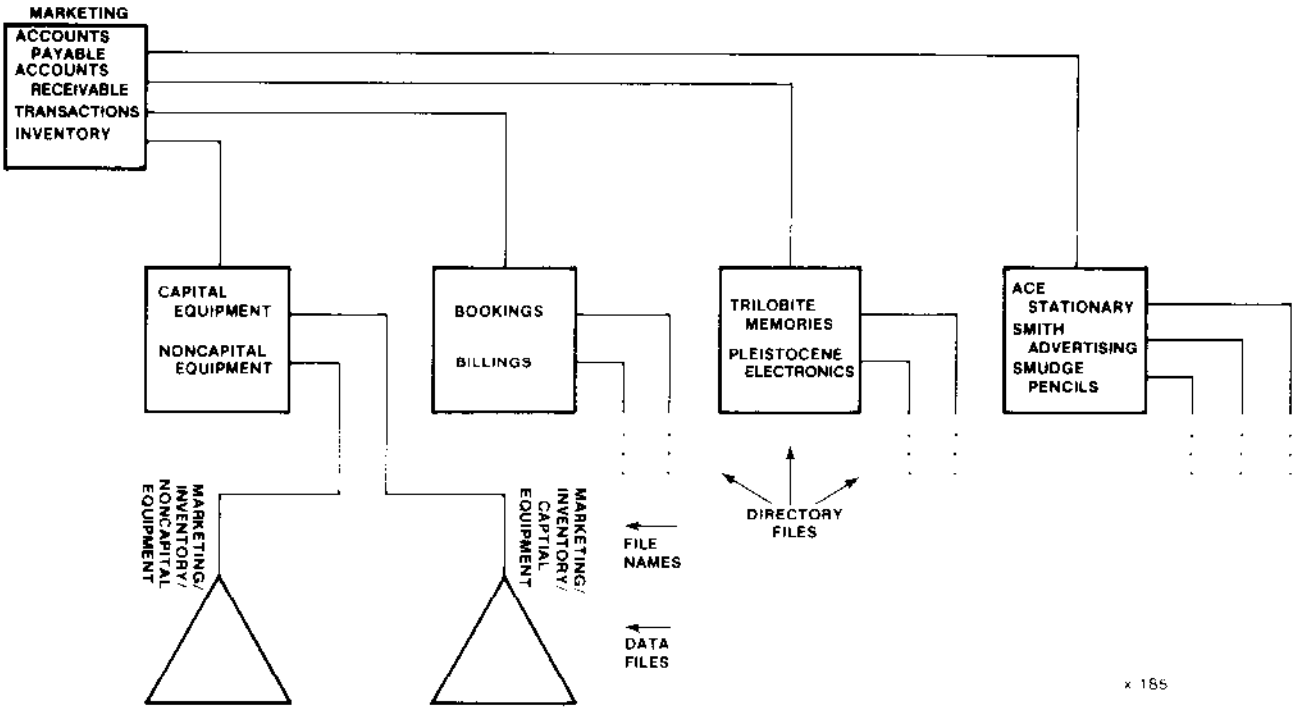


Figure 4-5. Hierarchical Naming of Files

Just as Figure 4-5 shows that single-level directory naming is not sufficient for all collections of files, another figure could be constructed to show that two-level directory naming is not always sufficient. Consequently, the iRMX II Operating System supports any number of levels of directories. This multi-level directory naming is called hierarchical naming of files.

Hierarchical naming of files simplifies the process of adding new applications to your system. One concern about expanding your system is the naming of mass storage files associated with a new application. Names of new files must differ from names of existing files. If your system uses only a few mass storage files, you can expect little difficulty in assigning unique file names. But if your system uses a large number of files, the problem of ensuring uniqueness becomes more significant.

This uniqueness problem becomes particularly difficult in a multi-user system. Hierarchical file naming eliminates the problem. Whenever you add a new user or application to your system, you can assign the user or application a directory. The new user or application can then use this directory to provide unique names to any number of files.

4.3.4 File Access Control

The iRMX II Operating System enables your application system to control access to hierarchically named files. This facilitates file sharing while still preventing valuable data from being copied, modified, or destroyed by unauthorized users.

In the multiprogramming environment provided by the iRMX II Operating System, the sharing of files can be useful. But the job that owns a file may wish to share it with only certain other jobs rather than all other jobs. Furthermore, the job owning a file may wish to restrict the nature of the shared access. For example, the owning job may wish to allow a particular file to be read but not written. The ability to specify how and with whom a file is shared is called file access control.

In the iRMX II Operating System, the owner of a file can specify who can use the file and how they can use it. In fact, the owner can even grant different combinations of access (reading only, writing only, reading and writing, etc.) to each user. Refer to the *Operator's Guide to the Extended iRMX II Human Interface* for more information on file access rights.

By controlling which jobs or who can access a file and how it is accessed, your system becomes more reliable in a real-time environment and more secure in a multi-user environment. Less risk exists that an unauthorized job or user accidentally modifies a valuable file or reads a confidential file.

Your application software can, in fact, expand file access protection into a file security system. For instance, suppose that your application involves several users accessing files on disk. By providing each user with a password, so an individual's identity can be verified, your application software can strictly control which users have what types of access to which files.

4.3.5 Control Over File Fragmentation

In the iRMX II Operating System, you can specify the granularity of each mass storage file. This control allows you to trade faster I/O for more efficient use of space on the mass storage device, or vice-versa.

When information is stored on a mass storage device, space is allocated in chunks rather than one byte at a time. These chunks, called granules, can be large or small, but all granules within one file must be the same size. This size is called the file granularity, and it is specified by the user who creates the file.

Within limits, a file's granularity affects file access time and file storage as follows:

- Sequential File Access — With this type of access, larger granularity sizes generally improve access time. Each access can handle more amounts of data.
- Random File Access — With this type of access, smaller granularity sizes generally improve access time. Each access handles less data that is not needed (less time is spent transferring needless data).
- Wasted Device Space — The file granularity directly affects the amount of wasted space on the device. More device space is wasted with larger granularity.

Here's an example. (For the sake of simplicity, we will ignore any data stored on the device needed for the operating system.) Consider a file containing 20,010 bytes. If the granularity is 10,000 bytes, the file occupies three granules, each of which is 10,000 bytes long. The first two granules are full and the third contains only 10 useful bytes. Although this file wastes 9,990 bytes of storage space, the data transfer rate is quicker than with a similar file of smaller granularity.

If we change the file granularity to 200 bytes, the file occupies 101 granules. Each of the first 100 granules is full, while the last granule contains only 10 useful bytes. The file now wastes only 190 bytes of storage space, but the data transfer rate is slower than the preceding example.

Since you can control granularity, you can trade device space for performance. If your application system has many mass storage units and space is readily available, you can specify a large file granularity. This gives you faster average transfer rates and shorter access times at the expense of device space.

If you have only one small mass storage unit, you might want to sacrifice some performance for better use of space. This consideration is especially true if you do not use the device often enough to notice the rate of data transfer.

4.3.6 Selection of Device Drivers

A device driver is a software module that serves as the interface between a device controller (which is essentially hardware and firmware) and the iRMX II Basic I/O System. The driver makes all devices look alike to the Basic I/O System. In effect, it hides the idiosyncrasies of a device from the Basic I/O System.

For use with the iRMX II Operating System, you can either select an Intel-supplied device driver (that is, one that supports Intel hardware) or create your own custom device driver.

By selecting and creating device drivers, you can attach any device to your application system. This means that you are not limited to a few specific devices. You can select devices on the basis performance, cost, reliability, availability, or whatever you choose.

4.3.7 Remote Files

In addition to supporting files on a number of secondary-storage devices, the iRMX II Operating System also supports the concept of remote files. With remote files, a user or task on one system can store or have access to files on another system. This access includes files resident on systems running an operating system other than iRMX. Remote files enable iRMX to be connected via OpenNET™ to XENIX-, MS-DOS-, VAX/VMS-, and iNDX™-based systems. The iRMX Networking Software package (available separately) provides the facilities for implementing remote file support.

4.3.8 Terminal Support Code

Even systems that do not include the Human Interface or the standard Command Line Interpreter (CLI) can have the ability to communicate with a wide variety of keyboard terminals. The iRMX II Terminal Support Code provides a programmable interface between a terminal driver and the Basic I/O System. This support code enables terminal operators to communicate with the system and gain access to a variety of special terminal modes and operations.

The major capabilities of the Terminal Support Code include:

- **Editing and Controlling Terminal Input** — A variety of characters available for controlling and editing terminal input exist. You can replace default control characters with different characters. You can also switch a terminal to "transparent mode," so that the use of editing and control characters has no effect on the input line.
- **Type-Ahead** — If you type faster than the operating system can read, interpret, and respond, the Terminal Support Code stores the data you type in a type-ahead buffer. The operating system then uses the data from this buffer when it is ready for it.
- **Controlling Terminal Output** — You can set the Terminal Support Code so that output sent to your terminal displays continuously, scrolls a few lines at a time, stops, or is completely discarded.

- Translation

  The Terminal Support Code accepts escape sequences (characters preceded by an ESC character) to define characteristics of a terminal. This feature enables you to "characterize" terminals so that the I/O system can use standard control codes and sequences of codes for all terminals. This process is called translation. Escape sequences are also used to set terminal variables, such as the number of lines displayed when in scrolling mode. You can change terminal behavior by keying in escape sequences or by running a program that sends the escape sequences.

4.4 CUSTOMIZING FEATURES

The iRMX II Operating System is designed specifically for Original Equipment Manufacturers (OEM) and Volume End User (VEU) applications. For this reason, the application system as a whole can appear unique to the user. Certain features of the operating system allow an application to be customized in its capabilities and in how it appears to the end user. The next few sections describe these features.

4.4.1 Custom Interactive Commands

Users interact with your applications by entering commands and receiving information. The iRMX II Operating System enables you to define commands that are meaningful to the operator and appropriate to the application. This command facility is called the Human Interface.

By designing commands appropriate to your users, you can control the human-to-application interface. This control can make a system appear "friendly," give the application a unique character, and reduce operator errors.

The first word in a command is the name of an executable program file on a mass storage device such as a disk. This facility gives you great flexibility in defining commands. When a user enters a command, the program having the command name is loaded from the secondary storage and is run by the operating system. This gives you the following advantages:

- You may add or modify commands simply by writing new programs.
- The number of custom commands for a system is not limited by the amount of dynamic memory.

- You do not have to "rebuild" the system to change commands.
- Programs used infrequently do not take up memory space when they are not being run.

4.4.1.1 Command Line Parsing

System calls are available for retrieving and interpreting parameters of a command. This process is called "parsing a command line."

Consider an application that monitors toxins in the blood of hospital patients. An operator (perhaps a nurse or doctor) can run a task that displays the toxin level of an individual patient, or of all the patients being monitored.

One approach would be to have the operator run the task using the following command:

The program might prompt with the following message:

```
Display of which units?--
```

In a more "friendly" approach, the operator could use commands oriented to the application and to his or her skills, rather than using computer-oriented commands. In the example, a better command might be as follows:

The program TOXIN issues a system call to receive the parameter "John Doe." Because file names are frequently parameters for commands, specialized system calls are also available to interpret file name parameters.

4.4.2 Application Loading

The iRMX II Application Loader gives a programmer great flexibility in the way programs use memory. The system can load programs anywhere in available memory, and programs can execute even though they are actually larger than the memory available.

The operating system enables your application to read programs from disk into memory and to run them. Also, the operating system allows a program to be broken into pieces called overlays, so that the entire program does not have to be in memory at one time. The following two sections describe the two methods in which program loading occurs:

4.4.2.1 Dynamic Loading

The loader modifies appropriate addresses in the program at the time the program is loaded. This capability, Dynamic Loading, offers great flexibility in the design of application systems. As new programs are added, existing programs do not have to be rebuilt ("linked") to run together. Or, if more memory is added to the system, the memory can be readily used.

4.4.2.2 Overlay Loading

Overlay loading is a design decision made by the programmer in the interest of conserving memory. If the programmer knows that, because of the logical flow of the program, two modules are mutually exclusive (that is, will never be required in memory at the same time), he may designate these modules as "overlays." The operating system will assign the same space in memory to both modules and load each module into that space when it is required.

Each overlay occupies the same area of memory but runs at a different time. A program containing overlays consists of a "root" that is always present in memory while the program is running, and of two or more overlays. The overlays are loaded when needed by system calls issued from the root.

An overlay facility allows programs to be run even if the programs are too large to fit in memory. Naturally, you must ensure that functions performed by separate overlays do not have to run simultaneously. Also, a program with overlays executes somewhat slower than one that does not contain overlays.

4.4.3 Simultaneous Multiple Terminal Support

More than one terminal can access the iRMX II Operating System at the same time. Two ways exist to implement this multiple-terminal support: the multi-user feature of the Human Interface and simultaneous multiple terminal support with I/O programs. The advantage of this flexibility is that more than one person at a time can access your system regardless of whether or not you employ the Human Interface.

4.4.3.1 Multi-User Feature of the Human Interface

The Human Interface layer of the operating system provides high-level multi-user support for multiple terminal communication. From a terminal in a multi-user system, an operator can execute commands, run development programs (e.g., editors, compilers, and linkers), and run application programs.

Here is how multi-user support works through the Human Interface. When the system is booted, the Human Interface initializes each terminal as either a static logon terminal or a dynamic logon terminal. The type depends on how the terminals were specified during configuration. If the terminal is "static," a specific user is always associated with that physical terminal. If the terminal is "dynamic," any valid user can log on and use the terminal.

After terminal initialization occurs, the operating system starts up an initial program. The initial program can be either the one that comes with the Human Interface or your own custom initial program.

Intel supplies an initial program called a Command Line Interpreter (CLI). The CLI is a program running under the iRMX II Human Interface that enables an operator to communicate with the operating system by entering commands at a terminal. As each command is entered, the CLI divides it into a program name and parameters, runs the program indicated by the command, and passes the parameters to the program. The following list describes the major attributes of the CLI:

- Support for Different Kinds of Terminals

  The CLI is designed to support various types of terminals. To determine the attributes of any user's terminal, the CLI examines a terminal specification file called :CONFIG:TERMCAP. In this file, each terminal type is given a name and assigned characteristics. Operators can edit this file to change the characteristics of a terminal or to add support for new terminals. The CLI provides a command called SET that enables operators to dynamically switch terminal types.

- Editing and Controlling Terminal Input

  The CLI is always ready to accept input from the terminal. You simply type characters at the keyboard and press Carriage Return (or some other Enter key that is defined in the TERMCAP file) to send the command.

  In addition, the CLI contains special function keys. These function keys enable you to move the cursor right or left within the command line, replace the current command line with a previous command line, execute a command line, delete characters within a command line, abort the current command, and continue a command onto the next line.

- Type-Ahead

  When you enter characters at the terminal, you automatically use the type-ahead feature to continuously enter command lines. The CLI sends the first line to the operating system for processing and saves additional data in a type-ahead buffer. After the operating system finishes with a line, the CLI fetches, displays, and begins processing the next line.

- Recalling Commands

  The CLI is capable of recalling the last 40 command lines entered. Using function keys, or the CLI command HISTORY, you can retrieve any of the previously entered lines that the CLI can recall. Once you retrieve a command line, you can edit the line to make any changes needed, and then execute the command line.

- Special CLI Commands

  When you enter a Human Interface command, or run a program that you have written, you enter the name of the command or program, plus any associated parameters. Normally, the command or program is then loaded into memory (usually from disk) and executed. The CLI also provides some commands of its own that allow you to perform special operations provided by the CLI. Some of these operations include the following:

  - Recalling command lines
  - Running commands in background mode, displaying a list of background jobs, and canceling background jobs
  - Assigning and canceling abbreviations (aliases) for commands
  - Altering the CLI environment
  - Invoking a set of commands in a batch file

- I/O Redirection

  The I/O redirection feature of the CLI allows programs that normally receive their input from the keyboard and transmit their output to the screen, to receive data from and send data to files or other I/O devices. This feature permits programs to run without user intervention, a condition that is especially useful when running in the background mode.

In addition to all of its features, you can expand the CLI to include user extensions. This characteristic enables you to add your own features and still retain the capabilities of the CLI.

You also have the option of providing your own initial program. This initial program might be a CLI of your own design, or it might be a completely different kind of program. For example, if you want a particular terminal to be used only for BASIC programs, you might design a BASIC interpreter to use as the initial program.

Multi-user support through the Human Interface is particularly versatile because you select, on a terminal-by-terminal basis, what initial program runs. For example, one terminal might run the Intel CLI, another might run a special CLI, and a third might always run a word-processing program.

4.4.3.2 Multiple Terminal Support with I/O Programs

You can implement multiple terminal support with your own programs. That is, you can replace the Human Interface iRMX II multi-user mechanism just described with your own programs.

In this case, your programs communicate with terminals through I/O system calls. You might do this if you need to implement functions not available with the Human Interface multi-user feature, or if you want to exclude the Human Interface layer from the operating system entirely. (A later section, "Interactive Configuration Utility," describes how you include and exclude layers of the iRMX II Operating System.)

4.4.4 Run-Time Binding

The iRMX II Operating System uses "run-time binding," the process of binding objects, files, devices, and application software with the tasks that use them. This binding method provides your system with three kinds of flexibility:

- Tasks in different jobs can share objects.
- Your procedures use logical names for files and devices.
- The process of attaching your application software to the iRMX II Operating System is simplified.

Before we look into run-time binding, let's consider binding as it relates to a program. Binding is the process of letting each program know the locations of the variables and procedures that it uses.

Binding can occur several times during the development and execution of a program. Some binding takes place during the process of compilation. For example, as a program compiles, the compiler assigns addresses relative to the beginning of the program to variables or procedures the program references. This process occurs whenever the compiler has sufficient information about a variable or procedure. Sometimes, however, a program refers to variables or procedures that are part of a separate program. When this happens, the compiler cannot resolve the reference, and binding must be delayed.

Some binding also takes place during linking. Linking is the process of combining several programs that have been compiled separately. The binding that occurs during linking allow a program to refer to variables and procedures defined in different programs. (Such references are called external references because they refer to information outside of the program.) When the linking process resolves an external reference, it performs binding that cannot be completed during compilation.

Run-time binding means binding while the system is actually running. The iRMX II Operating System provides three kinds of run-time binding:

- binding objects to tasks
- binding files and devices to tasks
- binding application software to the operating system

The first two kinds of run-time binding are based on the use of object directories. An object directory is an attribute of a job that allows tasks to provide ASCII names for objects. Tasks use the CATALOG OBJECT, LOOKUP OBJECT, and UNCATALOG OBJECT system calls to define, look up, or delete the name of an object. In each case, the task using the system call must specify the job whose object directory is to be accessed.

The next three sections expand on each type of run-time binding.

4.4.4.1 Binding Objects to Tasks

When two tasks use a mailbox to pass information, they must both access the same mailbox. But, if the programs for the two tasks are compiled and linked independently of one another (as they probably would be if they are in separate jobs), the tasks must use run-time binding to access the same mailbox.

The run-time binding of objects to tasks is accomplished as follows. When a task creates an object that it wishes to share with other tasks, the task that created the object catalogues the object in an object directory. Other tasks can then access the catalogued object if they know its ASCII name and its object directory.

Programmers can control the sharing of objects by selectively broadcasting object names. If two programmers wish to share an object, they must agree on both the name and the object directory that is to contain the name. One task then creates the object and the other accesses it through the object directory.

4.4.4.2 Binding Files and Devices to Tasks

Suppose you wish to code an application utility program that takes input from any supported input device or from a disk file. Run-time binding can help do this. The utility program can be coded to look up an input connection under a particular name. Then any program that needs the utility program can create the input connection, catalog it under the agreed-upon name, and invoke the utility program. In effect, the ASCII name in the object directory is the logical name of the input file.

4.4.4.3 Binding Application Software to the Operating System

Whenever your application software invokes a system call, an Intel-supplied interface routine redirects control to a protected-mode call gate. This call gate transfers program control to a procedure within the operating system that performs the desired function. In other words, the call gates bind the system calls of your application software to the iRMX II procedures.

Run-time binding provides your application system with flexibility. By allowing your system to name objects, the iRMX II Operating System provides a means of sharing dynamically created objects between jobs. By supporting logical names for files and devices, the iRMX II Operating System allows tasks to work with any combination of files and devices rather than with a single, fixed combination. By providing call gates to bind your application software to the operating system, you can reconfigure the operating system without having to recompile or relink your application software.

4.4.5 Error Handling

Error handling is the process of detecting and reacting to unexpected conditions. The iRMX II Operating System supports error handling by doing a large amount of validity testing and condition checking within system calls. Although a great amount of this type of checking occurs, the operating system cannot detect every error.

The iRMX II Operating System protects your system from most types of errors by using condition or exception codes and exception handlers.

- Condition Codes — Whenever a task invokes a system call, the iRMX II Operating System attempts to perform the requested function. Whether or not the attempt is successful, the operating system generates a condition code. This code gives you two pieces of information: it shows whether the system call succeeded or failed; and, in the case of failure, the code returned (an exception code) indicates which unexpected condition prevented successful completion.

- Exception Handlers — An exception handler is a procedure that the operating system can invoke when a task receives a condition code indicating failure of the function requested. As each task is created, it is assigned an exception handler, either one written by the programmer or a default exception handler provided by the operating system.

The alternative to using exception handlers is to process exception codes in the procedure that issued the system call.

Because you can write the exception handler, you can control the behavior of an application when it receives an exception code. For example, you can code the exception handler to recover from the error, delete the task containing the error, warn the operator of the error, or ignore the error altogether.

In summary, exception handling works as follows. The operating system generates a condition code for each system call. If the condition code indicates successful completion (the system call returns the mnemonic E$OK), the operating system detected no problems. If the condition code indicates a failure occurred, the exception code can be processed in either one of two ways: within the procedure that used the system call or by an exception handler invoked by the operating system. The technique used is a characteristic of a task and is established when the task is incorporated into the system.

Error handling provides your application system with two methods for reacting to unusual conditions. The first method, having the operating system automatically invoke your task's error handling procedure, greatly simplifies error processing. The second method, dealing with some or all of the unusual conditions within your application task, enables you to provide special processing for unusual circumstances.

4.4.6 Dynamic Memory Allocation

The iRMX II Operating System supports dynamic allocation of memory. Dynamic memory allocation reduces your implementation costs because you can build systems in which applications share memory. Also, your applications can change the amount of memory they use as their needs change.

Although there are numerous techniques for assigning memory to jobs, each technique falls into one of two classes: static memory allocation or dynamic memory allocation. In static memory allocation, memory is assigned to jobs when the system is starting up. Once the memory is allocated, it cannot be freed to be used by other jobs. Thus, the total memory requirement of the system is always the sum of the memory requirements of each job.

In dynamic memory allocation, jobs share memory. Memory is allocated to jobs only when tasks request it. When a job no longer needs the memory, one of its tasks can free the memory for use by other jobs. Dynamic allocation is also useful within a job. Some tasks can use additional memory to improve efficiency. An example of this is a task that allocates large buffers to speed up input and output operations.

Dynamic memory allocation provides your application system with reduced implementation costs. If your application system runs more than one application, chances are the memory demands for the various jobs will probably be out of phase. That is, one job will be freeing memory while another needs more. Dynamic memory allocation allows jobs to take advantage of this. Consequently, your application system requires less memory than it would when using static memory allocation.

4.4.7 Bootstrap Loading

The iRMX II Operating System contains a Bootstrap Loader that allows your application system to reside on disk and be loaded into RAM (random access memory).

A bootstrap loader is a program that resides in ROM on your application hardware. When your system is reset, the bootstrap loader receives control, and loads the rest of the software, including the iRMX II Operating System and the application software, into RAM.

The iRMX II Bootstrap Loader provides these advantages:

- By placing the iRMX II Bootstrap Loader in ROM, you can shift the rest of your application system to RAM. Since the rest of your system is probably much larger than the Bootstrap Loader, this shift decreases the amount of ROM required for your application.

  Keeping the rest of your application system in RAM decreases the amount of ROM required for your application. This savings in ROM space leads directly to reduced manufacturing costs. ROM, unlike RAM, requires that information be "burned" or masked into memory. By decreasing the amount of ROM in your system, the Bootstrap Loader reduces your masking or "burning" expenses.

- The iRMX II Bootstrap Loader simplifies the process of providing updated software to your customers. Rather than shipping ROM devices containing the modified software, you can ship diskettes, greatly reducing the cost of updating your software.

4.5 TOOLS

Along with the iRMX II Operating System, Intel provides software tools to help you develop an application system. Sometimes you will use the features listed in this section as part of your system, and sometimes you will use them only during development. Each feature simplifies the process of developing a complex system.

4.5.1 System Debugger

The iRMX II Operating System includes a System Debugger (SDB), which extends the capabilities of your system debug monitor (either the iSDM monitor or the D-MON386 monitor). The System Debugger provides "static" debugging facilities for those times when the system hangs or crashes, when you wish to "freeze" the system and examine it, or when synchronization requirements preclude the debugging of selected tasks. You can include the SDB as part of your application system during development, then remove it from your application when it has stabilized, thus reducing the size of the application system.

The SDB enables you to perform the following:

- identify and interpret iRMX II system calls
- display information about iRMX II objects
- examine a task's stack to determine system call history
- display information about the job hierarchy

The System Debugger provides the facilities necessary for diagnosing system crashes. By stopping the system, the SDB provides a global view of the system. Development time and costs are reduced because you can track down and fix errors quickly. For more information on the System Debugger, refer to the *Extended iRMX II System Debugger Reference Manual.*

4.5.2 SOFT-Scope® 286

The Soft-Scope 286 debugger is an interactive, source-level, symbolic debugging tool designed to accelerate software development. The Soft-Scope 286 debugger frees programmers from having to deal with the details of the CPU's machine code, or with the inner workings of the iRMX II Operating System. Features of the Soft-Scope 286 debugger include the following:

- automatic trapping of iRMX II protection exceptions
- source code interface and on-line listings
- access to program variables, including arrays and structures
- high-level break points
- disassembly of instructions
- second terminal option for remote debugging
- unlimited size of source files and number of symbols
- run-time exception handling
- iRMX multitasking support

- iRMX exception handling
- access to iRMX objects such as mailboxes and tasks
- ability to suspend and resume tasks

For more information on Soft-Scope 286, refer to the *Soft-Scope 286 Source-Level Software Debugger User's Guide.*

4.5.3 Start-Up Systems

The iRMX II Operating System is a configurable operating system, with pieces you can put together to create a custom system. However, the release package also contains versions of the iRMX II Operating System that are ready to use. The definition files used to create this system are also included. These ready-to-run systems are referred to as start-up systems.

The start-up system concept provides these specific advantages:

- Start-up systems can be used without modification on Intel's System 310 and 320 Microcomputers. No hardware or software changes are required to install the iRMX II Operating System on these systems.
- You can become familiar with the operating system immediately, and perhaps even run your application software without redefining or reconfiguring the operating system.
- The actual files used to create the start-up systems provide an example of how to put together an iRMX II system and may be used as a starting point for creating a custom system. Use these files to create the start-up system, then apply the appropriate iRMX II Update. Finally regenerate your system using the ICU generate (G) command. Refer to the *Guide to the Extended iRMX II Interactive Configuration Utility* for more information on generating a system.

4.5.4 Interactive Configuration Utility (ICU)

By selecting only the parts of the operating system you need, you reduce the amount of memory required for your application system. The Interactive Configuration Utility (ICU) supplies you with a straightforward method of choosing the parts of the operating system you want. This selection process is called configuration.

Two advantages to using Intel's ICU are as follows:

- Configuration of application systems, even complex systems, is relatively easy.
- The choices you make during the configuration process are saved in a file (definition file). You can later make changes to this file and reuse it. This technique allows for easy configuration changes based on an already configured system.

4.5.4.1 Configuration is Making Choices

As discussed in Chapter 1, the iRMX II Operating System consists of the following major subsystems or layers:

- Nucleus
- I/O Systems (Basic and Extended)
- Application Loader
- IHuman Interface
- System Debugger
- Universal Development Interface

To configure an application system based on the iRMX II Operating System, you first select the layers of the operating system that your application system requires.

After selecting which layers make up your system, you combine these layers with your application software, Intel-supplied software, and software from vendors. This combining forms the complete application system.

Finally, you install the application system on the target hardware.

Figure 4-6 illustrates the advantage of a configurable operating system. In the figure, an iRMX II Operating System consisting of the Nucleus, Basic I/O System, and Application Loader is combined with application software. By excluding unnecessary layers, you reduce the amount of memory your system needs.

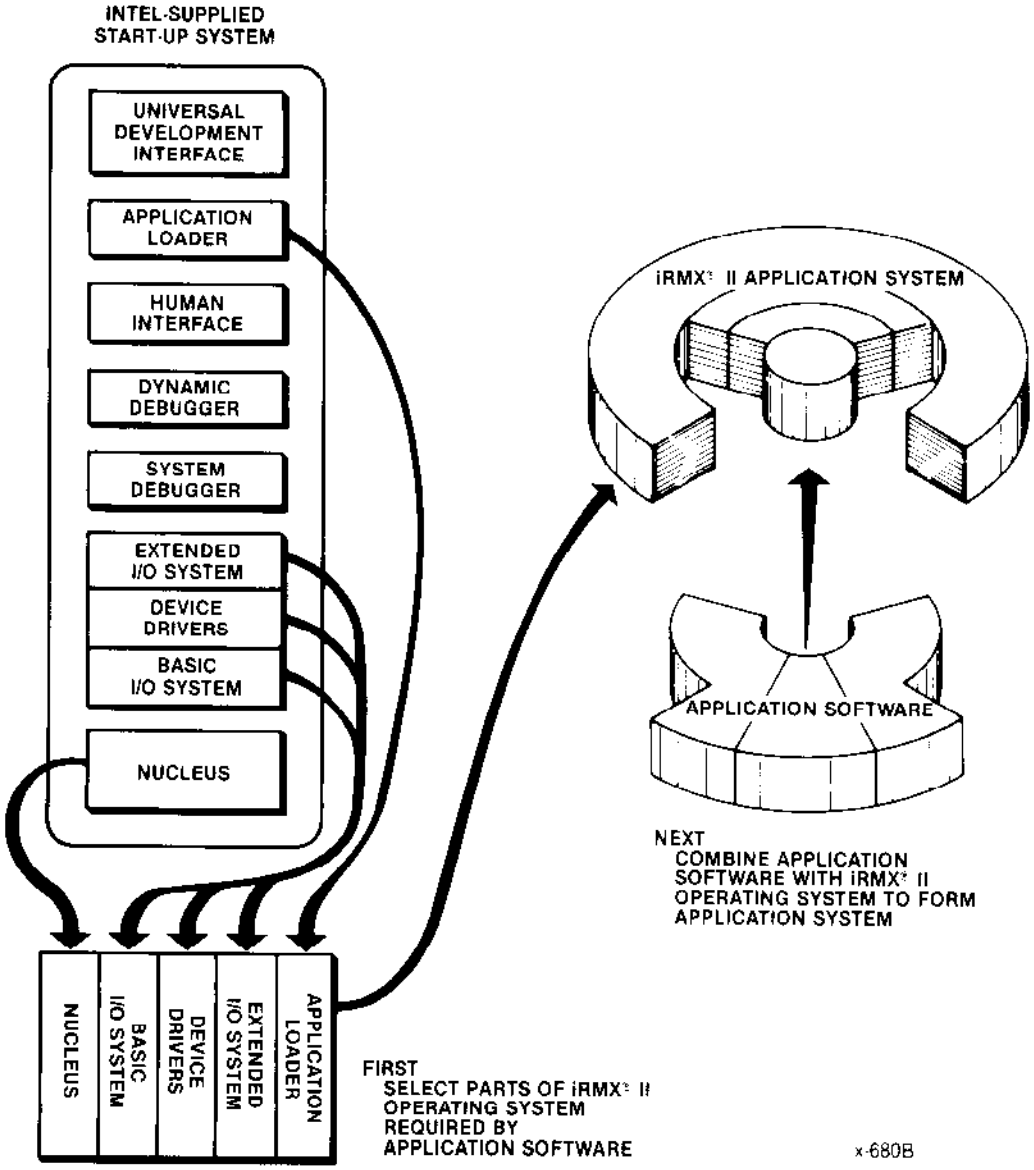


Figure 4-6. Configuration of an iRMX® II System

4.5.4.2 Configuration is Interactive

The Interactive Configuration Utility (ICU) guides you through the configuration process by displaying a series of "menus." Each menu describes a number of features. You can use the menu to accept or change an existing (or default) value for each feature. After you have made all your choices, you use the ICU generate command to generate your system. Generating the system produces a custom definition file that holds the information specific to the ICU session you are in.

4.5.5 File Maintenance Programs

As you develop an application, you need to work with files. You could write programs to perform these operations, but the operating system already has programs to perform operations usually necessary in developing an application system. These programs operate as commands to the system, as explained in the earlier section "Custom Interactive Commands."

Here is a list of some of the programs supplied with the operating system:

- ACCOUNTING — creates and modifies the file containing logon and logoff activities of all dynamic users in the system
- ADDLOC — integrates a data stream file image into an application system (used with the LOCDATA command)
- ATTACHDEVICE — attaches a physical device to the operating system
- ATTACHFILE — associates a logical name with an existing file
- BACKUP and RESTORE — saves and restores all of the files on a device
- COPY — copies or creates files
- CREATEDIR — creates a new file directory
- DATE — sets a new system date or displays the current date
- DEBUG — invokes the debugger to debug application jobs (used only if either the iSDM monitor or D-MON386 monitor is configured into the system)
- DELETE — removes data files and empty directories from secondary storage

• DETACHDEVICE	detaches a physical device from the operating system
• DETACHFILE	ends association of a logical name with a file
• DIR	displays a directory of the files on an iRMX II device
• DISKVERIFY	checks the data structures of iRMX II physical and named volumes
• DOWNCOPY and UPCOPY	moves files between Intellec® development system devices and iRMX II devices
• FORMAT	formats an iRMX II secondary storage device such as a disk or diskette
• INITSTATUS	displays the initialization status of Human Interface terminals
• JOBDELETE	deletes an interactive job
• LOCDATA	integrates a data stream file image into an application system (used with the ADDLOC command)
• LOCK	prevents users from logging onto a dynamic logon terminal
• LOGICALNAMES	lists all the current logical names available to the user
• LOGOFF	logs the user off a dynamic logon terminal
• MEMORY	displays the memory currently allocated to the user and the total system memory available to the user
• PASSWORD	adds or deletes users, or changes a logon password
• PATH	lists the pathname of a directory or file
• PAUSE	echos an optional message to the screen and waits for a <CR>
• PERMIT	sets user access to files
• RENAME	renames files

• RETENSION	Retensions a tape
• SHUTDOWN	provides the system with an orderly shutdown procedure
• SUBMIT	automatically executes a sequence of commands contained in an iRMX II file
• SUPER	changes the user identification number (ID) to the system manager ID
• TIME	sets the local or global system time, or displays the current time
• UNLOCK	enables terminals locked out of the system to log on
• VERSION	displays the version number of a file
• WHOAMI	lists the current user's identification and access rights
• ZSCAN	displays the identification number of all iRMX II fixes applied to that file

The most important advantage of these programs is that you save time and cost in developing your application system, because you already have the software tools necessary to manipulate files during the development process. Also, you can include the file maintenance programs as part of your application system if you need them.

4.6 ON-TARGET PROGRAM DEVELOPMENT

The iRMX II Operating System provides an ideal program development environment, allowing you to develop your programs on the same computer that will eventually run your code. The features of the operating system that make this on-target development work possible have already been described. Here is how they combine to provide a program development system:

• File Support	The iRMX II file system supports creation of source, object, and loadable files. Several programmers can use the same disk because of the hierarchical structure and protection mechanisms of the iRMX II file system.
• Command Line Interpreter	The CLI provides several commands that make on-target development more convenient. Included are commands to assign abbreviations for long command sequences, to run programs in background mode, to retrieve previously entered commands, and to run commands from batch files.

- Languages and Software tools

The Universal Development Interface is a standard flexible protocol that enables you to run various language translators (PASCAL-286, FORTRAN-286, PL/M-286, ASM286 Macro Assembler, etc.), language run-time packages, and other software development tools on the iRMX II Operating System.

The UDI protocol is a set of system calls by which language software uses the operating system. (Language software might be compilers, interpreters, assemblers, or run-time systems.) Any language may be run on the iRMX II Operating System as long as the language processor uses the UDI standard system calls and the Object Module Format (OMF) is compatible. In addition, the same language processor can, without modification, be run on any other operating system that includes the UDI system calls. (Intel markets a variety of operating systems that use UDI for language support.)

The UDI software interface gives you two major advantages:

-- A language processor can use well-defined, appropriate, standard calls to communicate with the iRMX II Operating System. Existing languages can be adapted easily to run on the operating system.

-- Any language processor or software tool (including user-written programs) using UDI system calls can run on several Intel operating systems. This feature, commonly termed "portability," is a major consideration in software design because it gains you economic benefits.

For the phases of program development, Intel provides the following software tools that run on the iRMX II Operating System:

-- Text editor (AEDIT)
-- Builder (BLD286)
-- Binder (BND286)

• Convenience	The Application Loader makes it easy to load and execute software. The Human Interface also provides a powerful facility for parsing the names of files used by language processors, editors, and linkers.
• Debugging	Programs developed on an iRMX II Operating System can be debugged using the iRMX II static System Debugger or Soft-Scope 286.

On-target program development using the iRMX II Operating System is useful for the following reasons:

- If your application system has spare resources (processing time, memory, mass-storage space), you can use the system more efficiently.
- Programmers can make changes on-site, which has economic and scheduling advantages.

If on-target development is not practical for the system you are developing, you can also develop an iRMX II application system on a host computer system and then transfer the system to a target computer. Intel Series IV Development Systems can be used as host systems for developing iRMX II code.

CHAPTER 5
A HYPOTHETICAL SYSTEM

5.1 INTRODUCTION

The previous chapter discussed the iRMX II Operating System features individually. This chapter revisits some of these features, now incorporated into a hypothetical system, to show you how features combine to form a powerful environment for your application software.

This hypothetical application system monitors and controls kidney machines in a hospital. These machines remove toxins from the blood of patients whose kidneys are not functioning correctly. The system, shown in Figure 5-1, consists of these main hardware components:

- Bedside Units

  One of these units is located at the side of each patient's bed. Each unit runs on an Intel iSBC® 286/12 processor board with the iRMX II Operating System. Each of these units performs four functions:

  -- measures the level of toxins in the blood as the blood enters the unit

  -- displays information so medical personnel at the bedside can monitor the dialysis process

  -- accepts commands from the bedside personnel

  -- removes toxins from the blood

  Each bedside unit performs these functions under the control of the central processing unit (CPU) on the processor board. That is, commands and measurements are sent to the CPU, which then adjusts the rate of dialysis and generates the bedside display.

- Master Control Unit — The system's master control unit (MCU) is an Intel System 320 Microcomputer, which consists of an iSBC 386/28 processor board, and a terminal with a screen and a keyboard. This system also runs the iRMX II Operating System. The MCU monitors the status of each bedside unit, enabling one person to monitor and control the entire system.
- iRMX-NET — This network connects the bedside units to the master control unit.

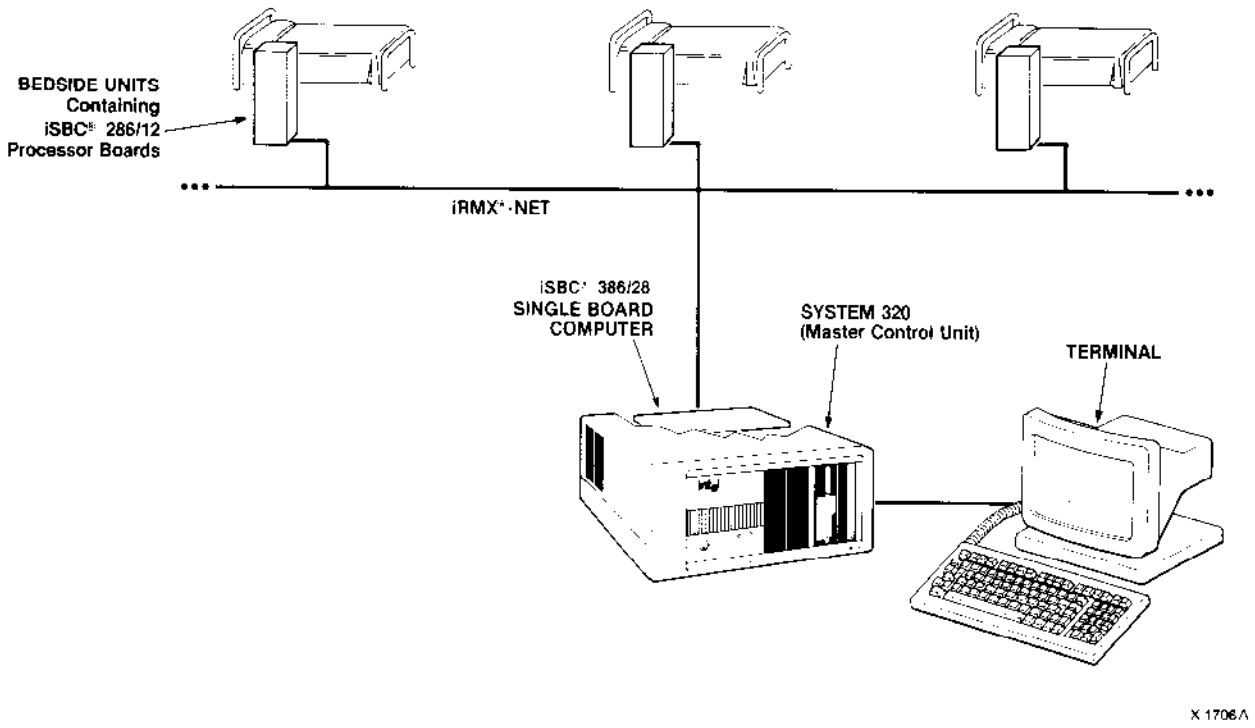


Figure 5-1. The Hardware of the Dialysis Application System

In summary, the system consists of software within the MCU's iSBC 386/28 processor board monitoring the entire system, and software within an iSBC 286/12 processor board controlling each bedside unit. Now, let's look at the software.

The application software controls the dialysis process by doing the following:

- obtains commands from the master control unit (MCU)
- obtains commands (if any) from each of the active bedside units
- reconciles the commands from the MCU and from the active bedside units
- obtains a toxicity level from each of the active bedside units
- creates a display at each active bedside unit
- creates a display at the MCU
- controls the rate of dialysis at each of the active bedside units

Now that we have roughly examined the nature of the system, let's investigate how the iRMX II Operating System fits in, starting with interrupt processing.

5.2 INTERRUPT PROCESSING

Three kinds of information flow from the bedside units to the MCU: toxicity levels, commands, and emergency signals when something has gone wrong.

The toxicity levels, measured as the blood enters the bedside unit, are not subject to abrupt change. The machine slowly removes toxins from the blood while the patient's body, even more slowly, puts toxins back in. The result is a steadily declining toxicity level.

Because of their nature, toxicity levels must be monitored regularly, but not too frequently. Let's suppose that the bedside unit computes the toxicity levels once every ten seconds then sends an interrupt signal to the MCU when the computation is complete.

The command information changes less predictably than the toxicity levels. Persons at the patient's bedside can enter commands through the bedside unit. Let's suppose that after encoding the information they press a button labeled ENTER that sends an interrupt signal to the MCU.

An emergency is not predictable and requires immediate attention. If the patient's toxic level begins to rise, the bedside unit sends a signal overriding all other interrupt signals to the MCU.

Each time the MCU receives an interrupt signal, the MCU stops executing the current instruction and begins to execute an interrupt task designed to handle the condition caused by the interrupt. In our example, when the MCU is interrupted by the ENTER signal, it begins running the interrupt task for bedside commands.

You must write the interrupt tasks for your system's custom devices, so the bedside-command task may serve as an example for you. The task performs the following steps:

- It checks a predetermined mailbox for a message from the MCU. The only task that waits at this mailbox is the task that reconciles bedside commands with the commands from the MCU.
- It puts the command information, along with the number of the bedside unit that received the command, into a message.
- It sends the message to the predetermined mailbox.
- It surrenders the CPU to the operating system.

One advantage of interrupt processing now becomes clear. Instead of wasting time polling the bedside units to see if a command has been issued, the application system can do other things until interrupted by the unit. When an interrupt (an event) does occur, it is quickly converted into a message and placed into a mailbox for processing by a task. The system then returns to its normal priority-based, pre-emptive scheduling. This technique enables your system to deliver more throughput.

Interrupt processing also provides the application system with flexibility. For instance, you can add more bedside units without modifying the system's software.

Finally, interrupt processing enables the system to respond immediately in applications demanding prompt responses to events.

5.3 HUMAN INTERFACE

Interaction between medical personnel and the system can be very smooth since the commands can be displayed in a form meaningful to the system operators. Also, new capabilities can be added to the system by simply adding new programs.

5.4 MULTITASKING

The entire application system is based on the multitasking capability of the iRMX II Operating System. Tasks are run using the pre-emptive, priority-based scheduling discussed in Chapter 4. This allows the more important tasks (such as those controlling the bedside units) to pre-empt lower-priority tasks.

5.5 INTER-TASK COORDINATION

The only form of intertask coordination used in our hypothetical dialysis system is intertask communication. The system uses a number of mailboxes to send information from one task to another. The simplicity of mailboxes enables engineers to divide the system into tasks on the basis of modularity rather than on minimizing intertask communication.

5.6 MULTIPROGRAMMING

Suppose that we extend the example to include statistical analysis in addition to dialysis. The two functions could advantageously be performed in different jobs. Why? Because they need to share very few resources.

If the statistical application has very little to do with the kidney application, they don't need to share mailboxes, tasks, or any other objects. Splitting them into two different jobs minimizes the chance that one application can affect the environment of the other.

But what if the two applications need to share only a little information? How can the shared data be passed from one job to another without losing the benefits of isolation? The iRMX II Operating System provides for this contingency in its implementation of run-time binding.

5.7 RUN-TIME BINDING

As mentioned earlier in this manual, run-time binding enables tasks of different jobs to share objects. As tasks create objects to be shared, the tasks catalogue the objects in an object directory. Then the tasks that need the objects can look them up by using their catalogued names.

Run-time binding also enables you to change the configuration of the iRMX II Operating System without recompiling or relinking your application software. For instance, suppose you have been including the iRMX II System Debugger in systems delivered to your customers. But now, a year or so after you started delivering systems, your product has stabilized: virtually no new bugs are being found. If you delete the SDB from your system, you can reduce the amount of memory required in any new systems you sell. The run-time binding of the system to your application software enables you to remove the SDB without making any changes to your application software.

5.8 MASS STORAGE FILES

As specified, the hypothetical system does not require mass storage files. However, a very reasonable extension of the current specification could include recording information about patients.

The iRMX II I/O System enables you to record information in files on flexible and hard disks, on tape, and in bubble memory. The system provides device handlers and disk formatting and allocating, and gives you a way to move information between main memory and the disk. Your application software need not include code to perform these functions.

If you added mass storage devices to the system, you could do program development on the system, so that new programs could be written and tested at the site. This is a powerful addition to a system, although it is not appropriate for every application.

5.9 DEVICE INDEPENDENCE

Even if the application system uses mass storage devices, device independence is not necessarily required. But, if the application is extended to allow the operator at the master control unit to send recorded data to any of several devices (e.g., teletypewriter, line printer, magnetic tape or disk), device independence becomes more important. The device-independent I/O System enables you to implement recording without adding code specific to each possible device.

CHAPTER 6
iRMX® II LITERATURE

6.1 INTRODUCTION

The iRMX II documentation set consists of five volumes organized into these five categories:

- introduction, installation, and operating information
- descriptions of each operating system layer and its use (User Guides)
- system calls
- system utilities and programming information
- configuration instructions

This chapter describes the contents of each volume and the manuals in that volume. The descriptions assume two types of programmers: system programmers and application programmers. System programmers configure the system, extend the operating system, write interrupt handlers, and perform other functions that affect the entire application system. Application programmers write application software.

This distinction is made because a system programmer's actions have a more global effect on the operating system and the application. Specifically, some system calls can, if used improperly, cause problems for all the tasks in the system; these calls should be used only by system programmers and, even then, only within operating system extensions. Other system calls affect only the task invoking the call.

Table 6-1 lists the contents of each volume of the documentation set.

Table 6-1. iRMX® II Volume Contents

VOLUME 1: *Introduction, Installation, and Operating Guides*
Introduction to the Extended iRMX II Operating System *Extended iRMX II Hardware and Software Installation Guide* *Operator's Guide to the Extended iRMX II Human Interface* *Master Index*
VOLUME 2: *Operating System User Guides*
Extended iRMX II Nucleus User's Guide *Extended iRMX II Basic I/O System User's Guide* *Extended iRMX II Extended I/O System User's Guide* *Extended iRMX II Human Interface User's Guide* *Extended iRMX II Application Loader User's Guide* *Extended iRMX II Universal Development Interface User's Guide* *Extended iRMX II Device Drivers User's Guide*
VOLUME 3: *System Calls*
Extended iRMX II Nucleus System Calls Reference Manual *Extended iRMX II Basic I/O System Calls Reference Manual* *Extended iRMX II Extended I/O System Calls Reference Manual* *Extended iRMX II Application Loader System Calls Reference Manual* *Extended iRMX II UDI System Calls Reference Manual* *Extended iRMX II Human Interface System Calls Reference Manual*
VOLUME 4: *Operating System Utilities*
Extended iRMX II System Debugger Reference Manual *Extended iRMX II Bootstrap Loader Reference Manual* *Extended iRMX II Programming Techniques Reference Manual* *Extended iRMX II Disk Verification Utility Reference Manual* *Guide to the Extended iRMX II Interactive Configuration Utility*
VOLUME 5: *Interactive Configuration Utility Reference*
Extended iRMX II Interactive Configuration Utility Reference Manual

6.2 VOLUME 1: INTRODUCTION, INSTALLATION, AND OPERATING INSTRUCTIONS

This volume contains three types of information: introductory material, hardware/software installation, and operating information.

Introduction To The Extended iRMX II Operating System -- This manual addresses the reader who is unfamiliar with the concepts and benefits of the iRMX II Operating System. Written at a less technical level than the other manuals, it introduces you to the iRMX II Operating System.

Extended iRMX II Hardware and Software Installation Guide -- This manual describes everything necessary to get the start-up systems installed on Intel System 300 Series Microcomputers. This includes instructions for installing wire jumpers on Intel single-board computers and device controllers, loading and running a start-up system, and using the Human Interface commands included in the start-up systems.

Operator's Guide to the Extended iRMX II Human Interface -- This manual describes the Command Line Interpreter (CLI) and Human Interface commands. The CLI provides commands that perform the following:

- show previously entered command lines
- provide aliases for command lines
- run programs in background mode

The Human Interface provides commands that perform functions such as the following:

- copy, delete, and otherwise manage files
- display directories
- format and verify mass storage volumes

The manual also describes the following subjects:

- file pathnames and other file information needed to use commands
- standard logical names
- the Human Interface

This manual also explains non-resident configuration--the process of adding new users to the system.

Master Index for the Extended iRMX II Operating System -- The Master Index is a complete by-subject index to the five-volume iRMX II documentation set.

6.3 VOLUME 2: OPERATING SYSTEM USER GUIDES

This introductory and operations-specific information is designed for first-time users.

Extended iRMX II Nucleus User's Guide -- Written for engineers planning to use the iRMX II Nucleus, this manual is the information warehouse for the Nucleus. It contains concise, yet detailed, discussions of these topics:

- the nature of objects in general and of tasks, jobs, semaphores, mailboxes, segments, and descriptors in particular
- task scheduling
- error processing
- interrupt processing
- the creation and deletion of extensions to the operating system
- region exchanges
- enabling and disabling the deletion of objects
- adding new types of objects to the operating system

Extended iRMX II Basic I/O System User's Guide -- This manual describes the iRMX II Basic I/O System and includes these topics:

- file directories and the types of files supported (named, stream, remote, and physical)
- user objects, and access rights associated with user objects
- I/O operations you may use with the operating system
- attaching and detaching devices
- system calls you may use in accessing the facilities of the Basic I/O System

Extended iRMX II Extended I/O System User's Guide -- This manual describes the iRMX II Extended I/O System. It covers the different types of iRMX II files. The Extended I/O System relieves programmers from the burden of details of I/O operations. In particular, the Extended I/O System data transfers are synchronous, meaning the operating system performs multiple-buffering operations, automatically synchronizing I/O operations with processing.

Extended iRMX II Human Interface User's Guide -- This manual describes the iRMX II Human Interface and includes these topics:

- a description of the command line interpreter (CLI), which provides line-editing features and commands for operations such as background and aliasing
- descriptions of Human Interface system calls used to parse custom commands, control programs run by the Human Interface, and send and receive messages to and from a terminal

- an explanation of the multi-user Human Interface, which lets the operating system communicate with many terminals simultaneously
- instructions for constructing command programs

Extended iRMX II Application Loader User's Guide -- This manual describes the Application Loader, which is used for two purposes:

- to load and run programs that reside on secondary storage (these programs are invoked by Human Interface commands)
- to load overlays by invoking system calls

Extended iRMX II Universal Development Interface User's Guide -- Designed for system and application programmers, this manual outlines general programming considerations for using the Universal Development Interface (UDI). The UDI is a software interface that allows language translators and other software development tools to access the facilities of the iRMX II Operating System. The manual also describes each of the UDI system calls that provide this access.

Extended iRMX II Device Drivers User's Guide -- This manual describes the device drivers contained in the iRMX II package and gives detailed instructions for writing a device driver that is compatible with the iRMX II I/O System. It also describes how to use the User Device Support (UDS) utility to add menus to the ICU to support user-written device drivers. Application programmers can use this manual to find out about existing drivers. System programmers can use this manual to add new devices to application systems.

6.4 VOLUME 3: SYSTEM CALLS

This volume describes all the system calls used within application programs. It details the structure and use of each system call and contains a list of the condition codes that the calls generate. (Since the contents of the manuals follow the same style and format, their contents are not listed here.)

- *Extended iRMX II Nucleus System Calls Reference Manual*
- *Extended iRMX II Basic I/O System Calls Reference Manual*
- *Extended iRMX II Extended I/O System Calls Reference Manual*
- *Extended iRMX II Application Loader System Calls Reference Manual*
- *Extended iRMX II Human Interface System Calls Reference Manual*
- *Extended iRMX II UDI System Calls Reference Manual*

6.5 VOLUME 4: OPERATING SYSTEM UTILITIES

This volume contains documentation for all the operating system utilities.

Extended iRMX II System Debugger Reference Manual -- This manual describes the System Debugger, a static debugging tool that enables you to diagnose system crashes and other "freeze" situations. The System Debugger is a memory-resident extension of the iSDM System Debug Monitor and D-MON386 debug monitor. This manual includes descriptions of System Debugger commands.

Extended iRMX II Bootstrap Loader Reference Manual -- This manual describes the iRMX II Bootstrap Loader, a program that can stand alone or be combined with the iSDM monitor. It is designed to load absolute code into memory.

Extended iRMX II Disk Verification Utility Reference Manual -- This manual documents the Disk Verification Utility, a software tool that runs as a Human Interface command, verifying and modifying the data structures of iRMX II named and physical volumes. The manual describes how to invoke the utility and contains detailed descriptions of all utility commands. Because users of the Disk Verification Utility must be familiar with the structure of iRMX II volumes, the manual describes iRMX II file and directory structures in detail. This manual also documents how to back up and restore volume file descriptor nodes.

Guide to the Extended iRMX II Interactive Configuration Utility -- The iRMX II Interactive Configuration Utility (ICU) leads a system programmer through configuration by displaying a series of "menus" or "screens" that describe each choice to be made. The programmer then selects the default answer or changes it. Using these answers, the ICU creates a file that automatically links and locates the application system software.

This manual describes the following:

- purpose of the ICU
- invocation of the ICU
- definition files
- editing commands
- ICU configuration screen formats
- upgrading a previous release of the operating system to the current release

Extended iRMX II Programming Techniques Reference Manual -- This manual provides system and application programmers with techniques to help save time and avoid common mistakes during system development.

6.6 VOLUME 5: INTERACTIVE CONFIGURATION UTILITY REFERENCE

This volume contains one manual: the *Extended iRMX II Interactive Configuration Utility Reference Manual.*

Extended iRMX II Interactive Configuration Utility Reference Manual -- This manual describes all the configuration parameters for every feature the ICU supports.

6.7. RELATED MANUALS

The manuals listed in this section are not contained in the iRMX II document set and must be ordered separately.

iRMX Networking Software User's Guide -- This manual describes how to implement iRMX-NET.

Each of the language manuals below is primarily a reference for use when you are writing and compiling (or assembling) programs in that language, but it also contains introductory information to familiarize you with the language.

- *PASCAL-286 User's Guide for iRMX 286 Systems*
- *PL/M-286 User's Guide for iRMX 286 Systems*
- *FORTRAN-286 User's Guide for iRMX 286 Systems*
- *ASM286 Assembly Language Reference Manual*

INDEX

E

F

H

I

J

L

M

N

O

P

R

S

T

U

intel

EXTENDED iRMX® II HARDWARE AND SOFTWARE INSTALLATION GUIDE

Intel Corporation
3065 Bowers Avenue
Santa Clara, California 95051

PREFACE

The Extended iRMX II Operating System is a software package that provides a real-time, multitasking environment for Intel sixteen bit single board computers and Intel-built microcomputers. This manual provides you with the information required to install your operating system on an Intel System 300 Series mircocomputer or a system that you build from Intel Single Board Computers.

The manual offers the following information:

- Describes the iRMX II Operating System Package you receive from Intel when you purchase the software.
- Steps you through the software installation procedure.
- Lists the information required to make the hardware modifications to support the extended iRMX II Operating System.

READER LEVEL

If you have an Intel System 300 Series microcomputer, then you do not need to know the following information.

If you are building your own system based on Intel single computer boards, you should be familiar with the following Intel software and hardware elements:

- The INTELLEC Microcomputer Development System and the iNDX Operating System.
- The ASM286 Macro Assembly Language and/or the PL/M-286 programming language.
- The iSDM System Debug Monitor.
- The individual hardware components that make up an extended iRMX II target system.

You may wish to read the *Introduction To The Extended iRMX II Operating System* manual.

NOTATIONAL CONVENTIONS

The following conventions are used in this manual:

<CR>	This symbol is used to represent a carriage return entered by the user.
<lowercase>	Fields appearing in lowercase in angle brackets (< >) indicate variable information. You must enter the appropriate value or symbol for variable fields. Information appearing in blue print indicates user input.

Intel®

CONTENTS

TABLES

FIGURES

CHAPTER 1
CONTENTS OF THE EXTENDED iRMX® II PACKAGE

1.1 INTRODUCTION

This manual provides you with instructions to install the Extended iRMX II Operating System software on Extended iRMX II-based or Series-IV Development Systems. It also gives you instructions on how to configure your hardware to run the Operating System if you are building a custom system.

This chapter describes the package of materials you receive from Intel when you purchase the iRMX II Operating System. The diskettes you receive with the iRMX II Operating System are called the Release Diskettes.

1.2 INTEL START-UP SYSTEMS

Intel provides you with four ready-to-run versions of the iRMX II Operating System, called Start-up Systems. The Start-up Systems are provided on diskettes called the Start-up System Boot Diskettes. You can bootstrap load the Operating System from the Start-up Boot Diskettes on the following Intel microcomputers: the System 286/310, the System 320, and MULTIBUS II MDP microcomputers. After booting, these versions of the Operating System are used to install iRMX II.

Bootstrap loading the iRMX II Operating System requires the use of two diskettes. One diskette contains only the bootloadable Operating System and the appropriate third stage. The second diskette contains all the system commands.

Bootstrap loading from the Start-up diskettes is described later in this manaul.

1.3 INVENTORY

Your shipment of the Extended iRMX II Operating System contains a set of manuals, one cartridge tape and a set of diskettes. Soft-scope, a source-level debugger, and the iRMX-NET Networking software, are sold separately.

1.3.1 Manuals

The set of iRMX II manuals is in five volumes. Each volume contains separate manuals that describe the Operating System. Table 1-1 lists the volumes and the manuals in each volume.

Table 1-1. Extended iRMX® II Operating System Volume Set

VOLUME 1 INSTALLATION AND PROGRAMMERS'S GUIDES
Introduction To The Extended iRMX II Operating System
Extended iRMX II Hardware And Software Installation Guide
Operator's Guide To The Extended iRMX II Human Interface
Master Index
VOLUME 2 USER GUIDES
Extended iRMX II Nucleus User's Guide
Extended iRMX II Basic I/O System User's Guide
Extended iRMX II Extended I/O System User's Guide
Extended iRMX II Human Interface User's Guide
Extended iRMX II Application Loader User's Guide
Extended iRMX II Universal Development Interface User's Guide
Extended iRMX II Device Drivers User's Guide
VOLUME 3 SYSTEM CALLS
Extended iRMX II Nucleus System Calls Reference Manual
Extended iRMX II Basic I/O System Calls Reference Manual
Extended iRMX II Extended I/O System Calls Reference Manual
Estended iRMX II Application Loader System Calls Reference Manual
Extended iRMX II Universal Development Interface System Calls Reference Manual
Extended iRMX II Human Interface System Calls Reference Manual
VOLUME 4 SYSTEM UTILITIES
Extended iRMX II System Debugger Reference Manual
Extended iRMX II Bootstrap Loader Reference Manual
Extended iRMX II Disk Verification Utility Reference Manual
Extended iRMX II Programming Techniques Reference Manual
Guide To The Extended iRMX II Interactive Configuration Utility
VOLUME 5 INTERACTIVE CONFIGURATION UTILITY REFERENCE MANUAL
Extended iRMX II Interactive Configuration Utility Reference Manual

1.3.2 Diskettes

The iRMX II Operating System is supplied on a set of twenty flexible diskettes. The format type available for the iRMX II Operating System is Double Sided/Double Density 5 1/4-inch iRMX-format soft-sectored diskettes. These diskettes can be read on an iRMX II-based microcomputer or on a SERIES-IV Development System. Table 1-2 lists the Release Diskettes that you receive from Intel.

1.3.3 Tape

The iRMX II Operating System is supplied on a single tape with the label "iRMX II Release 3 Operating System".

Table 1-2. Inventory of Extended iRMX® II Release Diskettes

Diskette Number	Diskette Name
Release Diskette Number 1:	iRMX II Start-up System Boot Diskette for System 286/310 Microcomputers
Release Diskette Number 2:	iRMX II Start-up System Boot Diskette for System 320 Microcomputers
Release Diskette Number 3:	iRMX II Start-up System Boot Diskette for iAPX 286-based MULTIBUS II Systems
Release Diskette Number 4:	iRMX II Start-up System Boot Diskette for iAPX 386-based MULTIBUS II Systems
Release Diskette Number 5:	iRMX II Start-up System Commands
Release Diskette Number 6:	iRMX II Nucleus and Communication Service
Release Diskette Number 7:	iRMX II BIOS, EIOS, and Application Loader
Release Diskette Number 8:	iRMX II Device Drivers and System Debugger
Release Diskette Number 9:	iRMX II Human Interface and UDI
Release Diskette Number 10:	iRMX II Include Files and Interface Libraries
Release Diskette Number 11:	iRMX II ICU Screen Master
Release Diskette Number 12:	iRMX II ICU Template and Definition Files
Release Diskette Number 13:	iRMX II ICU286 and ICU utilities
Release Diskette Number 14:	iRMX II Human Interface Commands, Diskette 1 of 3
Release Diskette Number 15:	iRMX II Human Interface Commands, Diskette 2 of 3
Release Diskette Number 16:	iRMX II Human Interface Commands, Diskette 3 of 3
Release Diskette Number 17:	iRMX Bootstrap Loader V7.0, Diskette 1 of 2
Release Diskette Number 18:	iRMX Bootstrap Loader V7.0, Diskette 2 of 2
Release Diskette Number 19:	iRMX II Demonstration Software
Release Diskette Number 20:	iRMX II ICU for Series-IV

1.4 RECOMMENDATIONS

To prevent the possibility of accidentally destroying system software, you should never remove the write-protect tabs on the release diskettes. When you have completed the software installation, be sure to store the release diskettes in a safe place.

CHAPTER 3
CARTRIDGE TAPE INSTALLATION OF iRMX® II.3 OPERATING SYSTEM FOR 80286- 80386-BASED SYSTEMS

3.1 INTRODUCTION

This document tells you how to use a cartridge tape to install the Extended iRMX II Operating System on 80286- and 80386-based microcomputers. **Please read this entire chapter before you begin the actual installation of the Operating System.** If you are installing from diskettes, refer to the next chapter for instructions.

Installing the Extended iRMX II Operating System means copying the contents of the iRMX II tape to your system. After you have completed the installation procedure outlined here, your system will be ready to:

- Develop and execute programs.
- Run the Interactive Configuration Utility (ICU).

3.2 THE EXTENDED iRMX® II SOFTWARE INSTALLATION

Installing the Extended iRMX II Operating System on your microcomputer involves the following:

- Backing up the files on your hard disk, and optionally retaining your existing iRMX files, if you have previously installed an operating system.
- Bootstrap loading the Operating System from the specially designated Start-up System diskettes.
- Executing an Intel-supplied SUBMIT file to prepare your hard disk.
- Installing the iRMX II Operating System.
- Installing the latest iRMX II Update package.
- Executing Intel-supplied SUBMIT files to copy the language diskettes onto your hard disk.
- Generating a version of the Operating System with the latest Update applied.
- Referring to other manuals before running your system.

3.3 PREPARING FOR SOFTWARE INSTALLATION

Before you install the Operating System, make certain that all the system hardware is working properly. Consult the Installation, Owner's Manual, and HARDWARE AND SOFTWARE INSTALLATION manual you received with your Intel microcomputer to ensure that the equipment is correctly set up and your terminal is connected with the correct cable before proceeding with this installation. Note that the terminal connected to the CPU board in the Intel microcomputer is referred to as the "system console." The system console is the terminal on which the monitor displays its output and is the only terminal from which you can bootstrap load the Operating System.

Once you have verified that the hardware is properly installed, inspect the diskettes you received from Intel to ensure that you have the proper number and the proper type. Table 1-2 lists the diskettes that you must have to install the Operating System.

One tape is required to install Extended iRMX II.3. It is labeled "Extended iRMX II.3 Operating System".

3.4 INSTALLING THE OPERATING SYSTEM

This section describes how to install the iRMX II Operating System onto your 80286- or 80386-based microcomputer using cartridge tape.

STEP 1: Backing Up Your Old Files

If you are installing the iRMX II Operating System onto a system for the first time (that is the system's hard disk has never been formatted) this step does not apply to you and you should proceed to Step 2 of the installation process.

This step applies if you have previously installed an operating system onto your microcomputer's hard disk and have created files of your own. Later in the installation process, you will have the option to format your microcomputer's hard disk. Whether or not you choose that option, we recommend that you use the Human Interface BACKUP command to save all of your files. This should be done now, using your present iRMX system, before starting the installation process. If you are unfamiliar with this command, refer to the *Operator's Guide to The Human Interface* for descriptions of both the BACKUP and RESTORE commands.

If you want to retain the ability to execute the iRMX II Release 2.0 Operating System after this installation is complete, refer to the section on Retaining an Older Version of the Operating System later in this chapter.

STEP 2: The System Confidence Test

The System Confidence Test (SCT) is a power-on diagnostic routine. There are three versions; one for System 310 Microcomputers, one for System 320 Microcomputers and one for the iRMX MDP. Refer to the section below which describes the system you are using.

SCT for System 310 Microcomputers

Turn on the power for your system. In about five seconds, a prompt will be displayed on the system console consisting of a single asterisk "*," a series of asterisks, a single lowercase "x," or a series of "x"'s depending on the version of the SCT your microcomputer uses. Within ten seconds of the display of any of these prompts, type in uppercase "U's" until the SCT begins to execute.

When the SCT starts executing, you will see status reports displayed on the system console. For specific information on the meaning of the reports, consult the Diagnostics or Owner's Manual supplied with your microcomputer.

After the SCT begins, it prompts you to enter a response to the question:

```
"Exit to iSDM after testing ?   Enter "y" or "n" [n]
```

After displaying the prompt, the SCT waits for your response. Respond with a to enter the iSDM monitor at the end of the execution of the SCT; don't try to bootstrap load the extended iRMX II Operating System yet.

Once the SCT has completed, the iSDM monitor will display

```
Interrupt 3 at <xxxx:yyyy>
.
```

where:

The period (".") is the prompt for the iSDM monitor.

<xxxx:yyyy> is the address where the entry into the monitor occurred.

At this point, you are ready to go on to Step 3.

SCT for System 320 Microcomputers

Turn on the power for your system. In about five seconds, a prompt will be displayed consisting of a series of "x's." Within ten seconds of the display of this prompt, type in uppercase "U's" until the SCT begins to execute.

When the SCT starts executing, you will see status reports displayed on the system console. For specific information on the meaning of the reports, consult the Owner's Manual supplied with your microcomputer.

After the SCT has tested memory and other hardware, the SCT prompts you to enter a response to the question:

```
Break to DMON-386 monitor (y or [n]) ?
```

Enter in response to this prompt.

or, the SCT prompts you to enter a response to the question:

```
Break to iSDM monitor (y or [n]) ?
```

Enter in response to this prompt. If you answered "n", the Bootstrap Loader would attempt to bootstrap load the default system from the hard disk.

At this point, the SCT has completed and it turns control over to the iSDM monitor which will display:

```
Interrupt 3 at <xxxx:yyyy>
```

The period (".") is the prompt for the iSDM monitor.

<xxxx:yyyy> is the address where the entry into the monitor occurred.

At this point, you are ready to go on to Step 3.

SCT for iRMX MDP Microcomputers

Turn on the power for your system. In about five seconds, a prompt will be displayed consisting of a series of "x's." Within ten seconds of the display of this prompt, type in uppercase "U's" until the SCT begins to execute.

When the SCT starts executing, you will see status reports displayed on the system console. For specific information on the meaning of the reports, consult the Owner's Manual supplied with your microcomputer.

After the SCT has tested memory and other hardware, the SCT prompts you to enter a response to the question:

```
Do you want to do more Testing or use DMON-386 ? Enter "y" or "n"[n] ?
```

Enter in response to this prompt.

At this point, the SCT has completed and it turns control over to the DMON-386 monitor which will display:

```
Interrupt 3 at <xxxx:yyyy>
>
```

where:

The greater-than symbol (">") is the prompt for the DMON-386 monitor.

<xxxx:yyyy> is the address where the entry into the monitor occurred.

At this point, you are ready to go on to Step 3.

STEP 3: Booting From The Start-up System Boot Diskette

Select the appropriate Start-up System Boot Diskette for your microcomputer from iRMX II.3 release diskettes Number 1, 2, 3, or 4. This is the diskette that you will use to bootstrap load iRMX II.

Insert the diskette so that the label is positioned toward the door handle of the diskette drive. The portion of the diskette on which the label is fastened is the last part of the diskette to be inserted.

Enter the following iSDM or DMON monitor command for your system to bootstrap load the Operating System:

```
.             (all systems except the two exceptions below)

.              (iSDM monitor with SCSI controller)

>              (DMON monitor with iSBC 186/224A controller)
```

The monitor "b" command instructs the Bootstrap Loader to load a file. When bootstrap loading, the diskette drive is referred to by the device name ":wf0:". When no pathname is specified, the default boot system "/SYSTEM/RMX86" is loaded. The invocation given above instructs the Bootstrap Loader to load the default boot system from the Start-up System Boot Diskette you selected.

On completion of the bootstrap loading process, the Extended iRMX II Start-up System has been loaded into memory and the system console displays the following instruction:

```
Insert the Start-up System Commands Diskette and type "G<cr>"

Interrupt 3 at <xxxx:yyyy>
.
```

or

```
Interrupt 3 at <xxxx:yyyy>
>
```

where:

<xxxx:yyyy>	is the address where the entry into the monitor occurred.
. or >	is the monitor prompt.

Remove the Start-up System Boot Diskette from the diskette drive and insert the release diskette Number 5 labeled "Start-up System Commands Diskette" diskette. Then enter your monitor's GO command, as requested on the system console, to complete the system initialization process:

```
.  <CR> (iSDM monitor)

>   <CR> (DMON monitor)
```

On completion of the Extended iRMX II system initialization process, the system console displays the following message:

```
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

                Extended iRMX II Release 3.0

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*


Logon:
```

Enter

to logon as the system manager. Now the system prompts you for the password by displaying the prompt:

```
Password:
```

Enter the default password for the system manager:

(This must be in all lowercase letters.)

Now the iRMX II Operating System will sign on with the banner:

```
iRMX II HI CLI, V3.0: USER=0
Copyright <years> Intel Corporation
-
```

Note that "super" is the name of the system manager. This is a special iRMX user whose user ID is 0. All installation of iRMX II files must be done while logged in as the system manager. The system manager has access to all files used during the installation process. This user also has Change Access rights to all iRMX files.

Next, the system prompts you for the correct date and time. Enter the date in any one of the following three formats:

```
month/date/year (e.g., 05/15/1987)
date month year (e.g., 15 JUN 1987)
date month year (e.g., 15 JUNE 1987)
```

After entering the date, the system echoes the information and prompts you for the time. Enter the time in the format HOURS:MINUTES:SECONDS (e.g., 15:20:00). You can omit the minutes and seconds fields; the system sets them to zero. The system responds by echoing the entered time.

After the date and time are entered and echoed, the system executes a command file named :prog:alias.csd. This file will define the following aliases for your convenience:

alias	command name
a	ALIAS
ad	:sd:sys286/attachdevice
adf	:sd:sys286/attachdevice wmf0 as :f:
aed	:lang:aedit
af	:sd:sys286/attachfile
bk	BACKGROUND
crdir	:sd:sys286/createdir
dd	:sd:sys286/detachdevice
df	:sd:sys286/detachfile
h	HISTORY
install	submit :config:cmd/instal(wmf0)
logs	:sd:sys286/logicalnames
ls	:sd:sys286/dir $ sort
lpr	BACKGROUND(100,100) copy #0 to :lp:
mksys	submit :config:cmd/mksys(#0)
pmw	:sd:sys286/permit #0 drau u=world
sh	:sd:sys286/shutdown w=0

iRMX II HI Command Line Interpreter (CLI) commands are indicated in upper case characters. After the aliases are echoed to the console, the system displays the lines:

```
END SUBMIT :prog:alias.CSD
END SUBMIT :prog:r?logon
```

The iRMX II system is now ready to execute your commands.

CHAPTER 2
THE EXTENDED iRMX® II DEVELOPMENT ENVIRONMENT

2.1 INTRODUCTION

The development of an Extended iRMX II-based application system requires several hardware and software components. Some of these components are always required and others are a function of the particular application system. Figures 2-1 and 2-2 show typical development hardware environments.

Figure 2-1 shows the easiest way of developing your application system. In this method, you develop your application on a System 300 Series or MDP Microcomputer. With such a microcomputer, your development and target systems are the same. This type of development environment is referred to as "On-target Development".

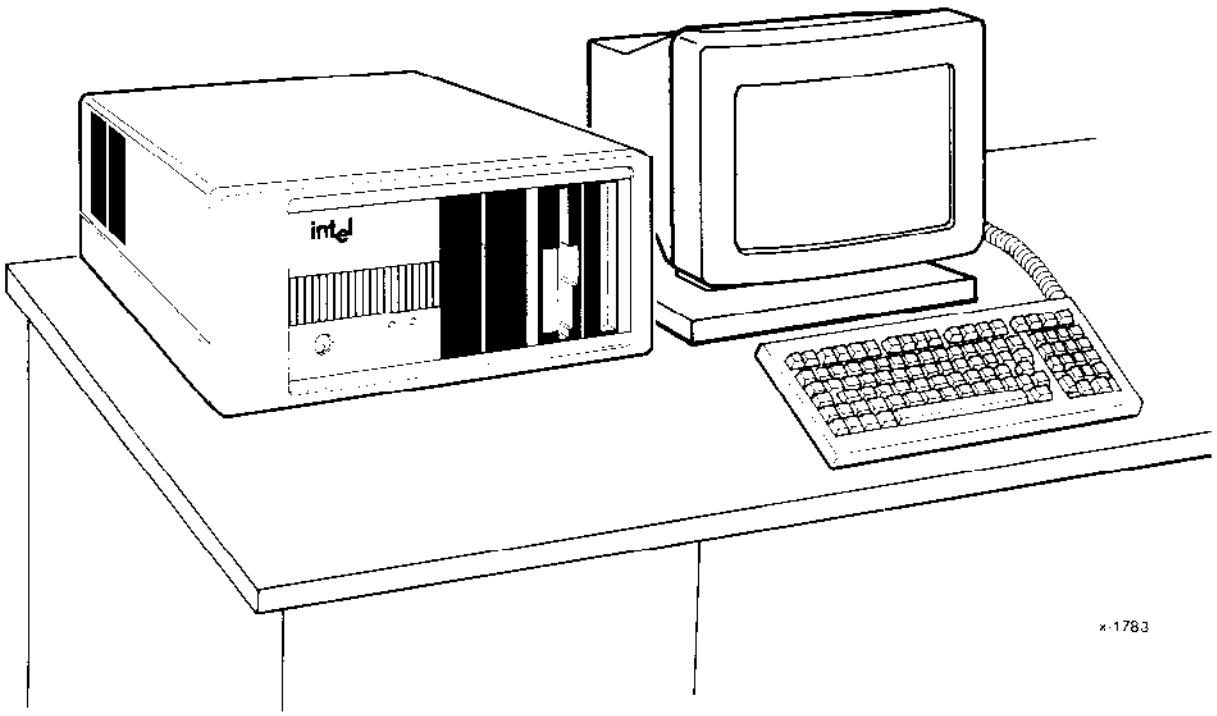


Figure 2-1. iRMX® II On-target Development Environment

Figure 2-2 illustrates a second way of developing your application system. A second iRMX II-based system or the Series-IV Development System is used to develop the application software. The application system must be down-loaded to the target system where the application actually runs. This type of development environment is referred to as "Cross Development". The figure shows devices that are commonly attached to the target system; however, you can also attach other available devices.

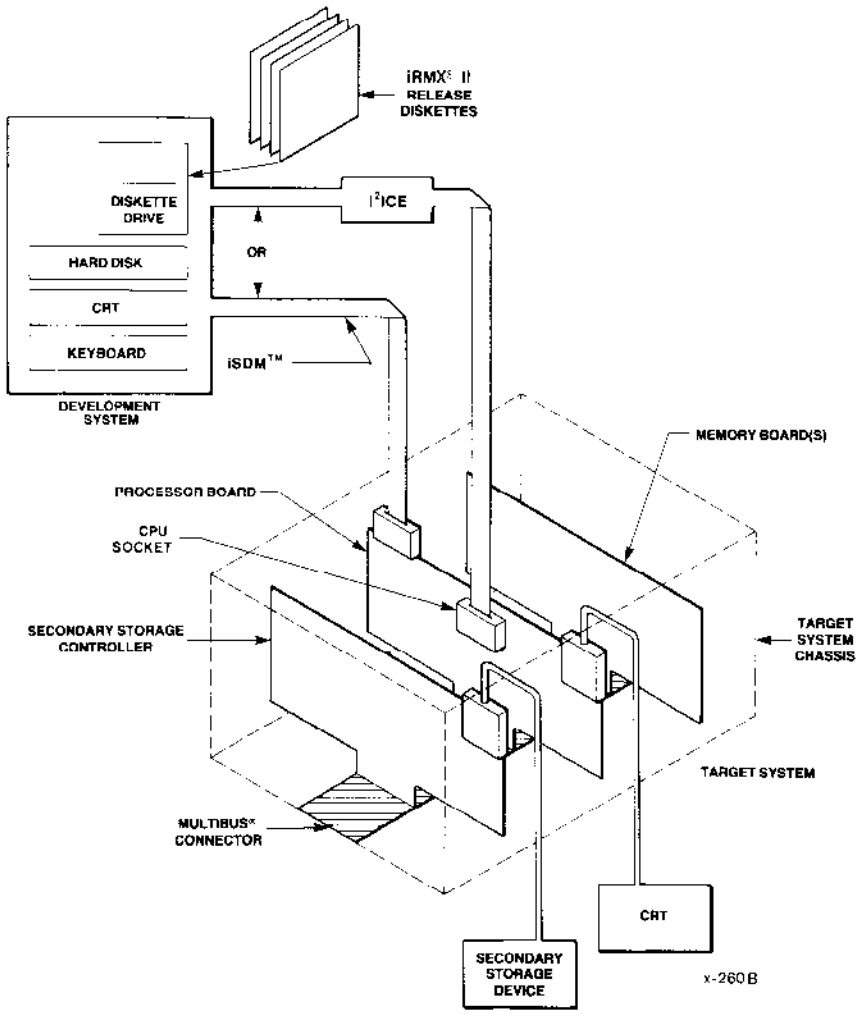


Figure 2-2. iRMX® II Cross Development Environment

2.2 GENERAL REQUIREMENTS

To develop your iRMX II-based application, you need one of the following environments:

- An iRMX II-based System 300 Series or MDP microcomputer (development and target system are in one package). A custom built iRMX II microcomputer may also be used. This type of development environment is called "On-target development".
- A development system and a separate target system. This type of development environment is called "Cross development".

These sets of equipment are discussed in the following sections.

2.3 AN iRMX® II SYSTEM AS YOUR DEVELOPMENT AND TARGET SYSTEM

One method of developing your iRMX II application system is to use a development system (the iRMX II Operating System) and a target system contained in one package. Intel provides the System 300 Series and the MDP microcomputers for on-target development. With Intel microcomputer systems, you can develop and test iRMX II application programs in an iRMX II development environment.

The Start-up Systems supplied with iRMX II are specifically designed to be run on System 300 Series and MDP MULTIBUS II microcomputers. This affords you the quickest method for starting the development of your application system.

Another method of developing your iRMX II-based application system is to build your own iRMX II-based system. Using the system you build, you can develop your application in an iRMX II Operating System environment. To use this method, you need the following equipment for your development system:

- An Intel single board computer (iSBC) based on the 80286 or 80386 microprocessor.
- A flexible diskette controller with at least one 5 1/4-inch drive.
- A hard disk drive and controller.
- A terminal connected to the serial line on your single board computer.
- A chassis/cardcage/power supply unit.
- The Bootstrap Loader in PROM and a monitor in PROM.
- At least one Megabyte of RAM.
- The iRMX II.3 Operating System and languages.

2.4 USING SEPARATE DEVELOPMENT AND TARGET SYSTEMS

A second method of developing your iRMX II-based application system is to use a development system on which to develop your software and a separate target system on which to run your application. To use this method, you must have the following software:

- The ASM286 Macro Assembler, the PL/M-286 compiler and the iAPX 286 Family Utilities.
- iRMX II Release Diskettes.

You must have one of the following development systems:

iRMX II-Based Development System

- The requirements are the same as were described earlier when your development and target systems are the same system.

INTELLEC Series-IV Development System

- INTELLEC Series-IV Development System with CRT, keyboard, at least one flexible diskette drive, one hard disk drive, and 128K-bytes of RAM.
- A diskette containing the iNDX Operating System (version 3.2 or later) for the Series IV.

Additionally, you must have a target system consisting of the following equipment:

- An 80286- or 80386-based microprocessor as the basic element of the application system.
- A chassis to supply power to the processor board(s) and any other system boards.
- Enough memory to contain the Nucleus, selected subsystems, and your application jobs. Most application systems require at least one Megabyte of RAM.
- If your application uses the Application Loader, the Human Interface, or the Bootstrap Loader, you must have secondary storage device(s) and appropriate controllers.

If you configure all of your software with an iRMX II-based Development System, you can use the following product to transfer code to the target system RAM for execution:

- The iSDM Monitor (Release 3.0 or newer.)

If you configure all of your software with a Series-IV Development System, you can use any of the following products to transfer code to the target system RAM for execution:

- The iSDM Monitor (Release 3.0 or newer.)
- The I^2ICE In-Circuit Emulator.

The I^2ICE In-Circuit Emulator and the iSDM Monitor transfer code from a secondary storage device on an Series-IV Development System, while the Bootstrap Loader, used in On-target development, transfers code from an iRMX II secondary storage device. The Bootstrap Loader is a much faster way to load the application system than either the I^2ICE In-Circuit Emulator or iSDM monitor.

After you have tested the code, you can burn it into PROM and place the PROM on the target system to eliminate using the I^2ICE emulator or the iSDM Monitor to load the code.

2.5 APPLICATION-DEPENDENT REQUIREMENTS

You may need additional hardware for your target system, depending on your application requirements. If your application includes an I/O System and you intend to use named or physical files, place at least one controller board in the chassis with the processor board. (Series 300 systems already contain the needed controller board(s).) You can use any of the following I/O and terminal controller boards:

I/O Controllers	Terminal Controllers
iSBC 214	iSBX 351
iSBC 215G/iSBX 218A/iSBX 217C	iSBX 354
iSBC 220	iSBC 534
iSBX 251	iSBC 544A
iSBC 264	iSBC 546
iSBC 208	iSBC 547
iSBC 186/224A	iSBC 548
iSBX 350 (line printer)	iSBC 188/48
	iSBC 188/56
	iSBC 186/410

NOTE

You can use controller boards other than those discussed in this section, but you must write the device drivers for them. The controller boards discussed in this section are the only ones for which Intel supplies device drivers.

Connect the controllers to their associated secondary storage devices. If only stream files are used, the I/O System can be used without a controller board.

Connect an RS-232C interface terminal to the serial I/O port of the processor board or terminal controller board if your application includes any of the following layers: the Basic I/0 System, the I/O System, or the Human Interface. One or more of the following terminal controller drivers must be configured into the Basic I/O System: 8251A, 8274, 82530, iSBC 534, iSBC 544A, , iSBC 186/410 or the Terminal Communications Controller (TCC) driver. The TCC driver is used with the iSBC 546, iSBC 547, iSBC 548, iSBC 188/48 and iSBC 188/56 terminal controller boards.

STEP 4: Installing the iRMX II Software

This step explains how to install the iRMX II software from cartridge tape to your system's hard disk. This step executes a SUBMIT file that performs all of the steps of installing the iRMX II files to your hard disk.

Two installation options are provided:

Option	Action
A.	Format your microcomputer's entire hard disk
B.	Format only Track 0 of the hard disk with the second stage of the iRMX II Bootstrap Loader

In the description of each option, you will read reasons for selecting that option. Before executing either option, insert the tape cartridge into the drive.

To do this, first remove the tape from the protective box. Next, grasp the tape cartridge by the end nearest the write-protect switch (labelled "SAFE"). Insert the tape into the tape drive with the write-protect switch nearest the drive's door handle. Push the tape cartridge in until it shifts slightly and does not eject when you release it. Move the drive's door handle toward the tape to the locked position.

OPTION A: Formatting Your Microcomputer's Entire Hard Disk Drive

You should choose this option if:

- You have a new microcomputer system that has nothing on the hard disk. All new Intel systems are shipped with unformatted (empty) hard disks.
- You wish to use the RESERVE feature of the Human Interface FORMAT command. This feature is used to create a copy of the maps that the iRMX II Operating System's uses to find keep track of your data. This copy is used if the working copy becomes corrupted. Corruption of a computer's file structure can be caused by power failure or other equipment failures.

  The Intel-supplied SUBMIT file that you execute to format a hard disk automatically specifies the RESERVE feature. Refer to the *Operator's Guide To The Extended iRMX II Human Interface* manual for details on the RESERVE feature of the FORMAT command.
- You have already backed up this disk and want to rebuild your file structure. The process of backing up your files, formatting the disk, and then restoring your files will improve the disk's file structure, which can become fragmented during normal usage. Fragmentation occurs when files are deleted, freeing space on various sectors; subsequently, when new files are created, this free space is used by placing parts of new files in them. This can cause the disk to spend a large amount of time seeking to the location where the next part of a file is located.

Step 1 of the Software Installation process contains information on how to backup your system.

The following Intel-supplied SUBMIT commands will format the entire hard disk. Type in the following command using Tables 3-1, 3-2, and 3-3 to fill in the three parameters.

where:

device	is the physical name of your hard disk. Refer to Table 3-2 for the appropriate physical device names for hard disks used in Intel microcomputers. Note that the device name is not a logical name so it does not have colons surrounding it.
interleave	varies depending on the type of controller and hard disk you are using. Refer to Table 3-1 for the appropriate interleave value.
files	is the maximum number of files you want to be able to create on your hard disk. This number depends on your application. A good rule of thumb is, if the microcomputer is to be used for development purposes, specify 125 files per megabyte of your hard disk's capacity. Thus, a 40 M-byte hard disk can reasonably be formatted to contain 5000 files. However, if the number of files needs to be changed later, you will have to backup your files and reformat your hard disk.

Table 3-1. Interleave Values

Controller	5 1/4-inch Peripherals	8-inch Peripherals
iSBC 214	Interleave is 1	–
iSBC 215G	Interleave is 4	Interleave is 5
iSBC 186/224A	Interleave is 1	–
SCSI Interface	Interleave is 4	–

Examples of Option A, Formatting the Entire Hard Disk

Remember! You must insert the cartridge tape before invoking this command.

A valid command to use for a microcomputer containing a Quantum Model Q540 hard disk configured as unit 0 controlled by the iSBC 186/224A or the iSBC 214 board would be:

```
SUBMIT :TAPE:FORM_DISK( QMA0, 1, 5000 ) <CR>
```

A valid command to use for a microcomputer containing a Maxtor Model XT-1140 hard disk configured as unit 0 controlled by the iSBC 186/224A or the iSBC 214 board would be:

```
SUBMIT :TAPE:FORM_DISK( MMA0, 1, 17500 ) <CR>
```

A valid command to use for a microcomputer containing a Toshiba Model MK56FB hard disk configured as unit 1 controlled by the iSBC 215G board would be:

```
SUBMIT :TAPE:FORM_DISK(TMA1, 4, 10625) <CR>
```

After the formatting of the hard disk is complete, the FORMAT command will issue a summary of the alternate tracks it assigned on the hard disk. Most hard disks contain bad tracks when they are delivered from the manufacturer so this is a normal condition, and assigning alternates is the method iRMX II uses to compensate for bad tracks. The message printed starts with the line:

```
The following tracks have been assigned an alternate:
```

.
.
{disk information}
.
.
.

Next, the DISKVERIFY command is executed to check the formatting of the hard disk. Following the DISKVERIFY command, the FORM_DISK.CSD SUBMIT file contains a series of the character 'y'. The 'y' is placed in the SUBMIT file to respond to any queries the DISKVERIFY command may issue. If the DISKVERIFY command does not make any queries, the error message 'illegal command' is issued which has no effect on the installation process. You should ignore this error message.

The iRMX II files will be added to your system from the tape automatically when the SUBMIT file invokes the Human Interface RESTORE command.

OPTION B: Formatting Only Track 0 of Your Hard Drive

This option formats only track zero of your system's hard disk and then places the second stage of the Bootstrap Loader on this track. No files on the disk will be affected by this option. You should choose this option if your disk was formatted using iRMX 286 Release 2.0 or iRMX 86 Release 7.0.

Note that you can also boot load the iRMX 286 Release 2.0 or iRMX 86 Release 7.0 Operating System from the same hard disk using the new second stage.

In selecting this option, you obtain the capabilities of the Bootstrap Loader's DEBUG switch. Refer to the *Extended iRMX II Bootstrap Loader Reference Manual* for details on the DEBUG switch.

Remember! You must insert the cartridge tape before invoking this following command.

To format only track 0 of your hard disk and install the Release 3.0 files, type the SUBMIT command below:

where:

device name — is the physical name of your microcomputer's hard disk. Refer to Table 3-2 for the appropriate physical device names for hard disks used in Intel microcomputers. Note that the device name is not a logical name so it does not have colons surrounding it.

This SUBMIT file will ensure the system manager has full access to the directories and files affected by the installation process. It will also rename the following iRMX 286 R3.0 directories and then will re-create them for Release 2 installation:

Release 2.0 Name	Renamed to
:SD:SYS286	:SD:SYS286_R2
:SD:RMX286	:SD:RMX286_R2
:SD:USER	:SD:USER_R2
:SD:BOOT	:SD:BOOT_R2

This is done as a precautionary measure to avoid inadvertantly modifying any of your existing files. A later section describes how to merge the contents of the renamed directories with the newly created directories.

Performing this option will cause the DISKVERIFY command to be invoked using the "FIX" option. The "FIX" option automatically upgrades the file structure by including accurate checksums for each file.

As the DISKVERIFY utility executes, it will correct discrepancies it finds on the hard disk. Refer to the *Extended iRMX II Disk Verification Utility Reference* manual for information on the meaning of these messages. If your hard disk was previously formatted using the iRMX 86 Release 7 or iRMX 286 Release 1 Operating System, this SUBMIT file will compute checksum values for the file structure on the disk. When this occurs, the following message, indicating that corrective action has been taken, will be repeatedly printed to the screen.

```
FILE=(file name, fnode):LEVEL=level:PARENT=parent:TYPE=type
Bad Checksum : value, Should Be : checksum ... FIXED
```

Following the DISKVERIFY command, the FORM_TRK0.CSD SUBMIT file contains a series of the character 'y'. The 'y' is placed in the SUBMIT file to respond to any queries the DISKVERIFY command may issue. If the DISKVERIFY command does not make any queries, the error message 'illegal command' is issued which has no effect on the installation process. You should ignore this error message.

This option saves the old version and temporarily creates new versions of the system configuration files :CONFIG:TERMCAP, :CONFIG:TERMINALS, :CONFIG:UDF and all the :CONFIG:USER/<user_name> user configuration files.

Your old iRMX files now exist in the /RMX286_R2 directory. You may wish to generate Release 2 versions of the Operating System. Refer to Retaining An Older Version of the Operating System for instructions on how to do this.

The iRMX II files will be added to your system from the tape automatically when the SUBMIT file invokes the Human Interface RESTORE command.

STEP 5: Copying The Boot System Onto The Hard Disk

The last command in the INSTALL file, from the previous step, switches the system device from being the tape drive to the hard disk and restarts the system. The system device is the device from which the Operating System reads its commands. When you invoked the SUBMIT file "FORM_DISK.CSD" or "FORM_TRK0.CSD", it was read from the Start-up System Commands diskette. Now the system will re-initialize and future commands will be read from the hard disk. You must again logon to the system specifying the name "super" and the password "passme" when prompted for them. You will be prompted to reset the date and time again. In any system that has a Global Clock (as on the iSBC 546 or MULTIBUS II iCSM boards) the date and time have not been set yet.

To set the DATE in the Global Clock, type:

```
date <date> global <CR>
```

where:

<date> has the form described in Step 3.

To set the TIME in the Global Clock, type:

```
time <time> global <CR>
```

where:

<time> has the form described in Step 3.

In this step you will install a bootable version of the Operating System on your hard disk. The default boot system will be changed during this installation. This file has the pathname "/SYSTEM/RMX86.286". You must either rename the default boot system (e.g. /system/rmx286r2.286) or rename the /SYSTEM directory. If you choose the first method, be sure to also rename the iRMX 286 third stage (e.g. /system/rmx286r2) to match the boot system name. If your boot system is in the /BOOT directory, it will be saved when the /BOOT directory is renamed.

Remove Release Diskette Number 5 and re-insert Diskette Number 1, 2, 3, or 4. Use the same diskette that you used to boot the system in Step 3. Enter the following command:

The command "INSTALL" is the alias for the HI command invocation "SUBMIT :CMD/CONFIG:INSTAL(WMF0)".

The parameter "WMF0" is the physical device name of the diskette drive and is appropriate for the System 310 and 320 microcomputers. Use "WMF0" on MDP systems using the iSBC 186/224A controller. If you are installing from a diskette that is not named "WMF0", you must issue the full command invocation, specifying the correct device name for the diskette drive. Use "SMF0" on systems using the SCSI interface.

The :CONFIG:CMD/INSTAL.CSD file attaches and detaches the diskette drive for you automatically. It is commonly used to install the contents of Release Diskettes onto the iRMX II system. It requires that the diskette whose contents are being installed contains a SUBMIT file named "INSTAL.CSD".

As the SUBMIT command copies the iRMX II boot file to the hard disk, a series of messages appear. If the system encounters an error during the process, it displays an error message but does not stop; the system will continue executing the SUBMIT command until it reaches the end of the command.

When the system displays an error message, stop the system by typing a CONTROL-C, detach the diskette drive by typing: DD :F:<CR> and correct the fault. For example, errors can be caused by inserting the diskette incorrectly. After entering the CONTROL-C, detach the diskette drive, remove the diskette, reinsert the diskette correctly, and enter the SUBMIT command again.

Also, if you are not logged on as the system manager, access rights to files may cause errors. After entering the CONTROL-C, detach the diskette drive and logoff. Then logon as the system manager, correct the access rights and enter the SUBMIT command again while logged on as the system manager. Remember, you should always install files on your iRMX II system while logged on as the system manager.

STEP 6: Installing The Language Utilities

The next step is to install the iRMX II Language Utilities, supplied in a separate box from the rest of the iRMX II Operating System.

Before beginning the installation of the language utilities, check that you have these diskettes:

1. iAPX286 BINDER AND LIBRARIAN FOR iRMX II-BASED SYSTEMS 1 OF 5
2. iAPX286 MAPPER AND OVERLAY GENERATOR FOR iRMX II-BASED SYSTEMS 2 OF 5
3. iAPX286 SYSTEM BUILDER FOR iRMX II-BASED SYSTEMS 3 OF 5
4. iAPX286 MACRO ASSEMBLER FOR iRMX II-BASED SYSTEMS 4 OF 5
5. 80287 SUPPORT LIBRARY FOR iRMX II-BASED SYSTEMS 5 OF 5
6. PL/M-286 COMPILER FOR iRMX II-BASED SYSTEMS
7. iAPX86 UTILITIES PACKAGE FOR iRMX II-BASED SYSTEMS 1/2
8. iAPX86 UTILITIES PACKAGE FOR iRMX II-BASED SYSTEMS 2/2
9. iAPX86 MACRO ASSEMBLER PACKAGE FOR iRMX II-BASED SYSTEMS
10. PL/M-86 COMPILER FOR iRMX II-BASED SYSTEMS
11. iRMX II AEDIT Text Editor

The Soft-Scope source-level debugger is supplied in a separate package. The installation procedure for the Soft-Scope debugger is described in the Soft-Scope manual.

To copy the iRMX II Language Utilities, insert each diskette into the diskette drive and enter the following command:

Where:

device name	is the physical name of the diskette drive. The parameter "WMF0" is the physical device name of the diskette drive and is appropriate for the System 310 and 320 microcomputers. Use "WMF0" on MDP systems using the iSBC 186/224A controller. If you are installing from a diskette that is not named "WMF0", you must issue the full command invocation, specifying the correct device name for the diskette drive. Use "SMF0" on systems using the SCSI interface.

This SUBMIT file will attach and detach the diskette drive for you automatically.

Again, if the system displays an error message, stop the system by typing a CONTROL-C, detach the diskette drive by typing "DD :F:" and correct the fault. Then re-invoke the SUBMIT file.

Once you have installed all the language diskettes, enter the following command:

This SUBMIT file copies the various libraries supplied with the languages you have just installed to the appropriate directories.

INSTALLING THE iC-286 COMPILER

To install the iC-286 compiler, Intel provides two SUBMIT files. To install the contents of the first iC-286 diskette, insert disk 1/2 and type:

To install the contents of the second iC-286 diskette, insert disk 2/2 and type:

INSTALLING THE FORTRAN-286 COMPILER

To install the Fortran-286 compiler, Intel provides two SUBMIT files. To install the contents of the first Fortran-286 diskette, insert disk 1/2 and type:

To install the contents of the second Fortran-286 diskette, insert disk 2/2 and type:

INSTALLING THE PASCAL-286 COMPILER

To install the Pascal-286 compiler, Intel provides three SUBMIT files. To install the contents of the first Pascal-286 diskette, insert disk 1/3 and type:

To install the contents of the second Pascal-286 diskette, insert disk 2/3 and type:

To install the contents of the third Pascal-286 diskette, insert disk 3/3 and type:

STEP 7. Preparing The AEDIT Editor For Use With Your System

If you use Intel's Aedit text editor, your system must contain a macro file named AEDIT.MAC. This macro file must be located in either the directory :LANG: (logical name for the directory which has the pathname /LANG286) or the directory :HOME:. The purpose of this macro is to define the attributes of your terminal to Aedit. Intel provides a number of macro files that at this point have been installed in the :LANG: directory. These macro files are named for the type of terminal that they define (For example, VT100.mac defines the VT100 terminal.)

Enter:

The system displays the files located in the :LANG: directory with the .MAC extension. Locate one that matches the type of your terminal. If you are not sure what terminal a macro defines, you can use the COPY command to print the macro on the system console. Each macro starts with a comment giving the full name of the terminal.

Once you have located the macro that defines your terminal, assuming you are still in SUPER mode from the previous step, type:

When this command completes, you can use AEDIT as the user "world". You should always be the system manager when you manipulate system files.

In cases where there is no macro that defines your terminal, first check to see if your terminal can emulate one of the terminals for which a macro is supplied. If this fails, you must write your own macro. Refer to the AEDIT manual included in the languages package for information on writing macros.

If you are installing the iRMX II Operating System on a microcomputer which will host multiple users (will be multi-user), you can copy the appropriate AEDIT macro file into each user's :HOME: directory. This will allow the AEDIT editor to be invoked by users who have different types of terminals. In this case, there should be no copy of AEDIT.MAC in the :LANG: directory.

STEP 8: Installing iRMX Networking Software

Intel provides the iRMX Networking Software product (iRMX-NET) for use in building Ethernet-based Local Area Networks (LANs). iRMX-NET allows extended iRMX II systems to share files with other extended iRMX II systems as well as iRMX 86, Xenix, iNDX, MS-DOS and VAX/VMS systems. If you are not installing the iRMX-NET Software, proceed to the next step.

If you have previously installed iRMX-NET and you chose the "Format Track Zero" option of Step 4 your iRMX-NET files have been saved. They now reside in a different directory than prior to this installation. This is a normal function of the "Format Track Zero" option of Step 4. You must install the iRMX II Update and execute the RMXMRG.CSD SUBMIT file described in the next step.

The information beyond this point is for users who are installing iRMX-NET for the first time or are installing a new release of iRMX-NET.

To install the iRMX-NET Networking Software, refer to the iRMX-NET Networking Software User's Guide which contains a description of the product and how to install it on System 300 Series Microcomputers. You can use the alias command 'INSTALL' to install the iRMX-NET files. After completing the iRMX-NET installation, return to the next step in this manual. Do not try to generate a version of the extended iRMX II Operating System which includes the iRMX-NET software at this time.

Note that the definition files supplied with iRMX-NET are very similar to those supplied with the iRMX II Operating System. But since they are released at different times, the iRMX-NET definition files may indicate that they do not match the version of the ICU when the ICU is invoked. This is normal,and you must use the ICU Restore feature to upgrade the definition files. (This Restore feature is not the HI RESTORE command.) Refer to the *Guide To The Extended iRMX II Interactive Configuration Utility* for details on the Restore feature of the ICU.

STEP 9. Installing The Update Package

An important phase of installing the iRMX II Operating System is the installation of the current iRMX II Update Package. You must perform this step even if you are installing a new release of the operating system. The update package is Intel's mechanism for fixing any software problems identified in the current version of the software. If you do not apply the update, you will be working with an outdated version of the iRMX II Operating System.

The Update Package accompanies all the shipments of the iRMX II Operating System. (The Update Package is shipped in a separate box.) Each Update consists of one or more Update Diskettes, an Update Installation Guide, Update change pages to the manual set and a customer letter.

The Update Diskette contains all of the fixes (ZAPs) to be applied to the iRMX II Operating System. The diskette is labeled:

"RMXII UP**x**R**y.z** "

where

x is the release level of the Update Package

y.z is the release level of the Operating System

The Update Installation Guide contains both detailed descriptions of each fix (ZAP) and detailed instructions on how to install the Update Package.

To install the Update to your system, find the Update Package, which is shipped in a separate box from the iRMX II Operating System, and follow the instructions in the Update Installation Guide. Also, make certain to read the Update Package Customer Letter for additional information on the Update Package.

STEP 10: Combining Directories when Upgrading to Release 3.0

If you invoked the FORM_DISK.CSD SUBMIT file in Step 4 (you formatted your entire hard disk), this step does not apply to you. Proceed to the next step.

Upgrading From iRMX® 286 Release 2.0

The following command combines the required files from iRMX 286 Release 2.0 into this release of the iRMX II Operating System. To perform this step, type the following command:

Upgrading From iRMX® 86 Release 7.0

If you are upgrading to iRMX II from iRMX 86 Release 7.0, type the following command to combine the required files from iRMX 86 directories into the iRMX II directory structure:

If you are upgrading to iRMX II from an earlier iRMX release, there are no files to merge and you should proceed to the next step.

STEP 11: Generating An Updated Version Of The Operating System

Now that you have installed the latest Update, you will generate an updated version of the Operating System. The version of the Operating System you loaded from the Start-up system Boot Diskette was generated from the original iRMX II libraries and, as such, does not contain any enhancements or corrections to problems. (Note that the version you generate in this step contains the System Debugger, the Start-up System does not).

Logoff from the system by typing:

```
-LOGOFF <CR>
```

The logon banner and prompt will be displayed. Now log back on to the system using the name "world" and a carriage-return for the password. It is not necessary to be the system manager to generate a configuration of the Operating System. You will be prompted to reset the date and time again. If your system contains a Global Clock, the date and time are still correct. Respond with "e" (EXIT) to leave the date and time at their current values.

Select an ICU definition file from those provided with iRMX II. Choose the default definition file which matches your microcomputer from those listed below.

ICU Definition File	CPU Boards Supported
28612.DEF	iSBC 286/10(A), iSBC 286/12
38620.DEF	iSBC 386/2x, iSBC 386/3x
286100A.DEF	iSBC 286/100A
386100.DEF	iSBC 386/116, iSBC 386/120

To generate the iRMX II Operating System, you will select the example commands that match your system. These examples follow the explanation below.

-
-
-

"boot_file_name" is the name of an ICU definition file **without the .DEF extension**. The ALIAS "CRDIR" is created for the HI command ":SYSTEM:CREATEDIR". Typing this alias will create a directory in which to generate an Updated system. The system generated is specified by the ICU definition file you select. The command "AF" is the alias for the HI command ":SYSTEM:ATTACHFILE". Typing these commands will create a directory for the system generation underneath your :HOME: directory and attach it as your current default directory.

The command "MKSYS" is the alias for the HI command invocation SUBMIT :CONFIG:CMD/MKSYS(#0). The #0 parameter in the MKSYS alias command receives the "boot_file_name". This convention of using the same name for the generation directory, definition file and created system is helpful if you generate several systems during the development of your application.

The Bootstrap Loader requires both a third stage and the Operating System itself in order to bootstrap load. The MKSYS command copies a third stage, which has a name that matches the "boot_file_name" used in the MKSYS invocation, from the directory /RMX286/BOOT to the directory /BOOT. The installation process places third stages with the following names in the directory /RMX286/BOOT: 28612, 38620, 286100A, SXM386, and 386100.

If you invoked the FORM_TRK0.CSD SUBMIT file in Step 4, you will probably have a directory in your :HOME: directory with a name identical to the one you will create next. Rename this directory now by typing:

An example of doing this, assuming that your system contains an iSBC 28612 processor board is:

```
RENAME 28612 to 28612_r2
```

If your processor board is the iSBC 286/10(A) or the iSBC 286/12 in a System 286/310 Microcomputer, select the 28612.DEF definition file and type:

```
-CRDIR 28612 <CR>
-AF 28612 <CR>
-MKSYS 28612 <CR>
```

This will create a new version of the iRMX II System named "/BOOT/28612.286". The MKSYS command will also make a copy of the Bootstrap Loader third stage appropriate for the new version of the Operating System named "/BOOT/28612".

If your processor board is the iSBC 386/2x or iSBC 386/3x, select the 38620.DEF definition file and type:

```
-CRDIR 38620 <CR>
-AF 38620 <CR>
-MKSYS 38620 <CR>
```

This will create a new version of the iRMX II System named "/BOOT/38620.286". The MKSYS command will also make a copy of the Bootstrap Loader third stage appropriate for the new version of the Operating System named "/BOOT/38620".

If your processor board is the iSBC 286/100A, select the 286100A.DEF definition file and type:

```
-CRDIR 286100A <CR>
-AF 286100A <CR>
-MKSYS 286100A <CR>
```

This will create a new version of the iRMX II System named "/BOOT/286100A.286". The MKSYS command will also make a copy of the Bootstrap Loader third stage appropriate for the new version of the Operating System named "/BOOT/286100A".

If your processor board is the iSBC 386/100/116/120 type:

```
-CRDIR 386100 <CR>
-AF 386100 <CR>
-MKSYS 386100 <CR>
```

This will create a new version of the System named "/BOOT/386100.286". The MKSYS command will also make a copy of the Bootstrap Loader third stage appropriate for the new version of the Operating System named "/BOOT/386100".

Execution of MKSYS may cause the main screen of the ICU to scroll several times. This is expected and does not indicate a problem.

After the MKSYS command completes, check the file, <boot_file_name>.out, for any errors that may have occurred during system generation. There are two methods of doing this check, you can use AEDIT if you installed it, or you can use the Human Interface COPY command. To use the COPY command, type:

```
-
```

You can use CONTROL-W to page through the file; each time you enter CONTROL-W your display will scroll one screen. Entering a CONTROL-Q exits this scrolling mode. CONTROL-S can also be used to halt the file during printing to the terminal, CONTROL-Q resumes printing to the terminal. Any errors indicate a problem. A complete listing of a generation output file is shown in the *Guide To The iRMX II Interactive Configuration Utility*.

Do not invoke the MKSYS command after you have successfully used it once to generate the Operating System. Instead, use the local copy of the definition file created for you by the MKSYS command. This version of the definition file can be the starting point to develop your custom application system.

You can create an ALIAS that makes invoking the ICU easier. To do this, use your editor to add the following line to your :PROG:ALIAS.CSD file:

To initialize the new ALIAS, type:

After doing this you can invoke the ICU by typing:

```
icu <pathname>/<definition file>.DEF <CR>
```

If you do not add this ALIAS, when you invoke the ICU, type:

```
/RMX286/ICU/ICU286  <pathname>/<definition file>.DEF <CR>
```

<pathname> is the optional directory pathname.

You can also use an ICU definition file that you created in Release 2.0 of the iRMX 286 Operating System with the Release 3.0 ICU.

STEP 12: Booting The Operating System From A Hard Disk

After the system has executed the SUBMIT command described in Step 11, the hard disk contains a version of the iRMX II Operating System suited to your CPU board. To test this version of the Operating System, you must shutdown the system properly by invoking the SHUTDOWN command. To do this, invoke the SUPER command and enter the password 'passme' when prompted. This causes you to become the system manager without logging on as the user "super". Type:

```
-
```

Respond with the password 'passme' to the password prompt. Now, type:

The command "SH" is the alias for the HI command invocation ":SYSTEM:SHUTDOWN W=0". After the SHUTDOWN command displays its shutdown complete message, reset the system by pressing the RESET button or turning the RESET switch on the front panel of your microcomputer.

The SCT, described earlier, starts executing, enter an uppercase U in response to the x's being printed on the screen. When your system console displays the monitor prompt, enter one of the following monitor commands to bootstrap load the newly created version of the Operating System:

```
.                                   iSDM Monitor

>                                    DMON-386 Monitor
```

where:

<boot_file_name> is the same name you used in Step 11 which reflects the particular CPU board in your microcomputer.

If you used the definition file, 28612.DEF, type: b /boot/28612

If you used the definition file 38620.DEF type: b /boot/38620

If you used the definition file SXM386.DEF type: b /boot/sxm386

If you used the definition file, 286100A.DEF, type: b /boot/286100A.

If you used the definition file, 386100.DEF, type: b /boot/386100.

Once the Operating System bootstrap loads and signs on, it indicates you have successfully generated a version of the Operating System. The logon banner and prompt will be displayed. Now log back on to the system using the name "world" and a carriage-return for the password. You will also be asked to set the date and time again. If you do not wish to do this again, respond with "e" when asked to enter the values.

Once you are satisfied with the new system, copy the new version of the system over the previous version. To do this, you must invoke the SUPER command and enter the password 'passme'. Now, type:

-

where:

<boot_file_name> is the same name you used in Step 11 to generate the Operating System.

If you do not specify a path name when entering the iSDM or DMON boot command, the Bootstrap Loader will load the default boot system. The pathname /SYSTEM/RMX86 is the default pathname for the Bootstrap Loader.

Note that a valid Bootstrap Loader third stage is provided in this step. When you create your own systems, you must remember to copy the correct third stage to the /BOOT path.

For a detailed description of the standard definition files, refer to Chapter 6.

STEP 13: Retaining an Older Version of the Operating System

This step is intended for users who chose the "Format Track Zero" option of Step 4. If you did not choose this option, skip this step. It explains how to move your renamed files to the directories where you want them. The following paragraphs explain the effects of the installation process on a iRMX II Release 2.0 system and an iRMX I Release 7.0 system.

An iRMX II Release 1.0 directory structure on the hard disk will be affected by the Release 3.0 installation process in these ways:

- The directory /USER will be re-created and will contain the logon file "R?LOGON" which is specific for iRMX II.3. It will execute a SUBMIT command named "ALIAS.CSD" which is not valid for an iRMX 286 R1.0 system but will have no adverse action if executed. If this is not acceptable, you should change the user's configuration file in the Release 1.0 configuration directory /CONFIG/USER/<user ID> to select the user's Release 1.0 renamed home directory. Later you can combine the files in the home directories.

An iRMX II Release 2.0 directory structure on the hard disk will be affected by the Release 3.0 installation process in these ways:

- The installation process renames the contents of the iRMX II Release 2.0 directories affected by this installation. The directories affected are: :SD:SYS286, :SD:RMX286 and :SD:BOOT. The FORM_TRK0.CSD SUBMIT file executed in Step 4 automatically renames them for you. The following names are the names used by the SUBMIT file:

Release 2.0 Name	Renamed to
:SD:SYS286	:SD:SYS286_R2
:SD:RMX286	:SD:RMX286_R2
:SD:BOOT	:SD:BOOT_R2

An iRMX 86 directory structure on the volume will be affected by this installation process in three ways:

1. The default boot system will be changed. This file has the pathname "/SYSTEM/RMX86". You must either rename the default boot system (e.g. /system/rmx86.86), make a copy of the boot system (e.g. copy /system/rmx86 to /boot86/rmx86) which you can use for booting, or you must rename the SYSTEM directory.
2. The directory /USER will be renamed to /USER_R2 and a new /USER directory will be created.
3. As an effect of change 2, the logon file "R?LOGON" will be specific for iRMX II. It will execute a SUBMIT command named "ALIAS.CSD" which is not valid for

iRMX 86 systems. If this is not acceptable, you should create a new home directory for the iRMX 86 user.

You can bootstrap load either Release 1.0, 2.0 or Extended iRMX II.3 versions of the Operating System, as well as the iRMX 86 Release 7.0 Operating System from the same hard disk after you have completed the installation instructions in this chapter.

You must use the Release 2.0 Interactive Configuration Utility (ICU) to modify the definition file that defines your Release 2.0 version of the Operating System and regenerate that version. You must modify three screens of the definition file which specify the pathnames of files the Operating System uses.

1. Change the pathname of the System Directory on the Human Interface screen using the SYS prompt to correspond to the new name of the SYSTEM directory (logical name :SYSTEM:):

   SYS=:SD:SYS286_R2

2. Change all the pathnames on the Includes and Libraries screen, except the Development Tools Prefix (the DTF prompt), to correspond to the new name of the Release 2.0 RMX286 directory. The following example shows how you can change the pathnames:

```
UDF=/RMX286_R2/UDI/
HIF=/RMX286_R2/HI/
        .
        .
        .
```

3. Change the pathname of the boot system, using the RAF prompt on the Generation Pathnames screen, to reflect the new pathname for the BOOT directory:

```
RAF=/BOOT_R2/<boot file name>.286
```

Now regenerate your Release 2.0 system by executing the SUBMIT file created by the ICU.

3.5 SYSTEM MAINTENANCE

From now on, whenever you logon to the system, Intel recommends that you do not logon as the system manager (user 'super') for security reasons. Instead, logon as either the user 'world' or as some other user that you have defined. Then, when system files need manipulation, invoke the SUPER command to temporarily become the system manager. This will afford significant protection from inadvertent deletion or modification of system files. Also, for system security, you should change the system manager password, using the PASSWORD command, as soon as you complete the installation process.

If problems occur, follow these steps:

- Reboot from the Start-up System Boot Diskette.
- Fix the problems on the hard disk.
- Reboot from the hard disk.

3.6 WHERE TO LOOK FOR INFORMATION TO GET STARTED

Once you have completed the steps listed in the previous sections, the iRMX II Operating System is fully installed and ready to use. Refer to the following manuals for additional help:

- For basic information about your system and the manuals in your documentation set, refer to the *Introduction To The Extended iRMX II Operating System.*
- For information about the commands that you can run from a terminal, refer to the *Operator's Guide to the Extended iRMX II Human Interface.*
- For an example program that discusses iRMX programming concepts refer to Appendix A of the *Extended iRMX II Programming Techniques Reference Manual* in Volume 4. In order to use these files, create a directory, and copy the contents of the /RMX286/DEMO/PLM/INTRO directory into it. Use these copies of the files and keep the ones in the original directory as backup copies. The following commands will accomplish this:

  -
  -
  -
  -

  For example programs that discuss MULTIBUS II objects and programming, refer to the *Extended iRMX II Nucleus User's Guide* in Volume 2.
- For information on how to add users to your system refer to the *Operator's Guide to the iRMX II Human Interface* .
- For information about memory partition sizes and further insights into the Installations (including information on how to generate a custom version of the

Operating System), refer to Chapter 5 of this manual, the *Guide To Using The Extended iRMX II Interactive Configuration Utility* and to the *Extended iRMX II Interactive Configuration Utility Reference Manual.*

CHAPTER 4
DISKETTE INSTALLATION OF THE iRMX® II.3 OPERATING SYSTEM FOR 80286- 80386-BASED SYSTEMS

4.1 INTRODUCTION

This document tells you how use the iRMX II diskettes to install the Extended iRMX II Operating System on 80286- and 80386-based microcomputers. **Please read the entire guide before you begin the actual installation of the Operating System.** If you are installing from a cartridge tape, refer to Chapter 3 for instructions.

Installing the Extended iRMX II Operating System means copying the contents of the iRMX II Release Diskettes to your system. After you have completed the installation procedure outlined here, your system will be ready to:

- Develop and execute programs.
- Run the Interactive Configuration Utility (ICU).

4.2 THE EXTENDED iRMX® II SOFTWARE INSTALLATION

Installing the Extended iRMX II Operating System on your microcomputer involves the following:

- Backing up the files on your hard disk, and optionally retaining your existing iRMX files, if you have previously installed an operating system.
- Bootstrap loading the Operating System from the specially designated Start-up System diskettes.
- Executing an Intel-supplied SUBMIT file to prepare your hard disk.
- Installing the iRMX II Operating System.
- Installing the latest iRMX II Update package.
- Executing Intel-supplied SUBMIT files to copy the language diskettes onto your hard disk.
- Generating a version of the Operating System with the latest Update applied.
- Referring to other manuals before running your system.

4.3 PREPARING FOR SOFTWARE INSTALLATION

Before you install the Operating System, make certain that all the system hardware is working properly. Consult the Installation, Owner's Manual, and HARDWARE AND SOFTWARE INSTALLATION manual you received with your Intel microcomputer to ensure that the equipment is correctly set up and your terminal is connected with the correct cable before proceeding with this installation. Note that the terminal connected to the CPU board in the Intel microcomputer is referred to as the "system console." The system console is the terminal on which the monitor displays its output and is the only terminal from which you can bootstrap load the Operating System.

Once you have verified that the hardware is properly installed, inspect the diskettes you received from Intel to ensure that you have the proper number and the proper type. Table 1-2 lists the diskettes that you must have to install the Operating System.

4.4 INSTALLING THE OPERATING SYSTEM

This section describes how to install the iRMX II Operating System onto your 80286- or 80386-based microcomputer using the release diskettes.

STEP 1: Backing Up Your Old Files

If you are installing the iRMX II Operating System onto a system for the first time (that is the system's hard disk has never been formatted) this step does not apply to you and you should proceed to Step 2 of the installation process.

This step applies if you have previously installed an operating system onto your microcomputer's hard disk and have created files of your own. Later in the installation process, you will have the option to format your microcomputer's hard disk. Whether or not you choose that option, we recommend that you use the Human Interface BACKUP command to save all of your files. This should be done now, using your present iRMX system, before starting the installation process. If you are unfamiliar with this command, refer to the *Operator's Guide to The Human Interface* for descriptions of both the BACKUP and RESTORE commands.

If you want to retain the ability to execute the iRMX II Release 2.0 Operating System after this installation is complete, refer to the section on Retaining an Older Version of the Operating System later in this chapter.

STEP 2: The System Confidence Test

The System Confidence Test (SCT) is a power-on diagnostic routine. There are three versions; one for System 310 Microcomputers, one for System 320 Microcomputers and one for the iRMX MDP. Refer to the section below which describes the system you are using.

SCT for System 310 Microcomputers

Turn on the power for your system. In about five seconds, a prompt will be displayed on the system console consisting of a single asterisk "*," a series of asterisks, a single lowercase "x," or a series of "x"'s depending on the version of the SCT your microcomputer uses. Within ten seconds of the display of any of these prompts, type in uppercase "U's" until the SCT begins to execute.

When the SCT starts executing, you will see status reports displayed on the system console. For specific information on the meaning of the reports, consult the Diagnostics or Owner's Manual supplied with your microcomputer.

After the SCT begins, it prompts you to enter a response to the question:

```
"Exit to iSDM after testing ?   Enter "y" or "n" [n]
```

After displaying the prompt, the SCT waits for your response. Respond with a to enter the iSDM monitor at the end of the execution of the SCT; don't try to bootstrap load the extended iRMX II Operating System yet.

Once the SCT has completed, the iSDM monitor will display

```
Interrupt 3 at <xxxx:yyyy>
.
```

where:

The period (".") is the prompt for the iSDM monitor.

<xxxx:yyyy> is the address where the entry into the monitor occurred.

At this point, you are ready to go on to Step 3.

SCT for System 320 Microcomputers

Turn on the power for your system. In about five seconds, a prompt will be displayed consisting of a series of "x's." Within ten seconds of the display of this prompt, type in uppercase "U's" until the SCT begins to execute.

When the SCT starts executing, you will see status reports displayed on the system console. For specific information on the meaning of the reports, consult the Owner's Manual supplied with your microcomputer.

After the SCT has tested memory and other hardware, the SCT prompts you to enter a response to one of the following two questions (depending on your monitor):

```
Break to DMON-386 (y or [n]) ?
```

Enter in response to this prompt.

or, the SCT prompts you to enter a response to the question:

```
Break to iSDM monitor (y or [n]) ?
```

Enter in response to this prompt. If you answered n, the Bootstrap Loader would attempt to bootstrap load the default system from the hard disk.

At this point, the SCT has completed and it turns control over to the iSDM monitor which will display:

```
Interrupt 3 at <xxxx:yyyy>
```

where:

The period (".") is the prompt for the iSDM monitor.

<xxxx:yyyy> is the address where the entry into the monitor occurred.

At this point, you are ready to go on to Step 3.

SCT for iRMX MDP Microcomputers

Turn on the power for your system. In about five seconds, a prompt will be displayed consisting of a series of "x's." Within ten seconds of the display of this prompt, type in uppercase "U's" until the SCT begins to execute.

When the SCT starts executing, you will see status reports displayed on the system console. For specific information on the meaning of the reports, consult the Owner's Manual supplied with your microcomputer.

After the SCT has tested memory and other hardware, the SCT prompts you to enter a response to the question:

```
Do you want to do more Testing or use DMON-386 ? Enter "y" or "n" [n]
```

Enter in response to this prompt.

At this point, the SCT has completed and it turns control over to the DMON-386 monitor which will display:

```
Interrupt 3 at <xxxx:yyyy>
>
```

where:

The greater-than bracket (">") is the prompt for the DMON-386 monitor.

<xxxx:yyyy> is the address where the entry into the monitor occurred.

At this point, you are ready to go on to Step 3.

STEP 3: Booting From The Start-up System Boot Diskette

Select the appropriate Start-up System Boot Diskette for your microcomputer from iRMX II.3 release diskettes Number 1, 2, 3, or 4. This is the diskette that you will use to bootstrap load iRMX II.

Insert the diskette so that the label is positioned toward the door handle of the diskette drive. The portion of the diskette on which the label is fastened is the last part of the diskette to be inserted.

Enter the following iSDM or DMON monitor command for your system to bootstrap load the Operating System:

```
.              (all systems except the two exceptions below)
.               (iSDM monitor with SCSI controller)
>               (DMON monitor with iSBC 186/224A controller)
```

The monitor "b" command instructs the Bootstrap Loader to load a file. When bootstrap loading, the diskette drive is referred to by the device name ":wf0:". When no pathname is specified, the default boot system "/SYSTEM/RMX86" is loaded. The invocation given above loads the default boot system from the Start-up System Boot Diskette you selected.

On completion of the bootstrap loading process, the Extended iRMX II Start-up System has been loaded into memory and the system console displays the following instruction:

Insert the Start-up System Commands Diskette and type "G<cr>"

```
Interrupt 3 at <xxxx:yyyy>
.
```

or

```
Interrupt 3 at <xxxx:yyyy>
>
```

where:

<xxxx:yyyy>	is the address where the entry into the monitor occurred.
. or >	is the monitor prompt.

Remove the Start-up System Boot Diskette from the diskette drive and insert the release diskette Number 5 labeled "Start-up System Commands Diskette" diskette. Then enter your monitor's GO command, as requested on the system console, to complete the system initialization process:

```
.       (iSDM monitor)
>        (DMON monitor)
```

On completion of the iRMX II system initialization process, the system console displays the following message:

```
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

                Extended iRMX II Release 3.0

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*


Logon:
```

Enter

to logon as the system manager. Now the system prompts you for the password by displaying the prompt:

```
Password:
```

Enter the default password for the system manager:

(This must be in all lowercase letters.)

Now the iRMX II Operating System will sign on with the banner:

```
iRMX II HI CLI, V3.0: USER=0
Copyright <years> Intel Corporation
-
```

Note that "super" is the name of the system manager. This is a special iRMX user whose user ID is 0. All installation of iRMX II files must be done while logged in as the system manager. The system manager has access to all files used during the installation process. This user also has Change Access rights to all iRMX files.

Next, the system prompts you for the correct date and time. Enter the date in any one of the following three formats:

```
month/date/year (e.g., 05/15/1987)
date month year (e.g., 15 JUN 1987)
date month year (e.g., 15 JUNE 1987)
```

After entering the date, the system echoes the information and prompts you for the time. Enter the time in the format HOURS:MINUTES:SECONDS (e.g., 15:20:00). You can omit the minutes and seconds fields; the system sets them to zero. The system responds by echoing the entered time.

After the date and time are entered and echoed, the system executes a command file named :prog:alias.csd. This file will define the following aliases for your convenience:

alias	command name
a	ALIAS
ad	:sd:sys286/attachdevice
adf	:sd:sys286/attachdevice wmf0 as :f:
aed	:lang:aedit
af	:sd:sys286/attachfile
bk	BACKGROUND
crdir	:sd:sys286/createdir
dd	:sd:sys286/detachdevice
df	:sd:sys286/detachfile
h	HISTORY
install	submit :config:cmd/instal(wmf0)
logs	:sd:sys286/logicalnames
ls	:sd:sys286/dir $ sort
lpr	BACKGROUND(100,100) copy #0 to :lp:
mksys	submit :config:cmd/mksys(#0)
pmw	:sd:sys286/permit #0 drau u=world
sh	:sd:sys286/shutdown w=0

iRMX II HI Command Line Interpreter (CLI) commands are indicated in upper case characters. After the aliases are echoed to the console, the system displays the lines:

```
END SUBMIT :prog:alias.CSD
END SUBMIT :prog:r?logon
```

The iRMX II system is now ready to execute your commands.

STEP 4: Preparing the Hard Disk

This step explains how to install the iRMX II software from diskettes to your system's hard disk. Two installation options are provided:

Option	Action
A.	Format your microcomputer's entire hard disk
B.	Format only Track 0 of the hard disk with the second stage of the iRMX II Bootstrap Loader

In the description of each option, you will read reasons for selecting that option.

OPTION A: Formatting Your Microcomputer's Entire Hard Disk Drive

You should choose this option if:

- You have a new microcomputer system that has nothing on the hard disk. All new Intel systems are shipped with unformatted (empty) hard disks.
- You wish to use the RESERVE feature of the Human Interface FORMAT command. This feature is used to create a copy of the maps that the iRMX II Operating System's uses to find keep track of your data. This copy is used if the working copy becomes corrupted. Corruption of a computer's file structure can be caused by power failure or other equipment failures.

  The Intel-supplied SUBMIT file that you execute to format a hard disk automatically specifies the RESERVE feature. Refer to the *Operator's Guide To The Extended iRMX II Human Interface* manual for details on the RESERVE feature of the FORMAT command.
- You have already backed up this disk and want to rebuild your file structure. The process of backing up your files, formatting the disk, and then restoring your files will improve the disk's file structure, which can become fragmented during normal usage. Fragmentation occurs when files are deleted, freeing space on various sectors; subsequently, when new files are created, this free space is used by placing parts of new files in them. This can cause the disk to spend a large amount of time seeking to the location where the next part of a file is located.

  Step 1 of the Software Installation process contains information on how to backup your system.

The following Intel-supplied SUBMIT commands will format the entire hard disk. Type in the following command using Tables 4-1, 4-2, and 4-3 to fill in the three parameters.

where:

device	is the physical name of your hard disk. Refer to Table 4-2 for the appropriate physical device names for hard disks used in Intel microcomputers. Note that the device name is not a logical name so it does not have colons surrounding it.
interleave	varies depending on the type of controller and hard disk you are using. Refer to Table 4-1 for the appropriate interleave value.
files	is the maximum number of files you want to be able to create on your hard disk. This number depends on your application. A good rule of thumb is, if the microcomputer is to be used for development purposes, specify 125 files per megabyte of your hard disk's capacity. Thus, a 40 M-byte hard disk can reasonably be formatted to contain 5000 files. However, if the number of files needs to be changed later, you will have to backup your files and reformat your hard disk. Refer to Table 4-3 for the number of files.

Table 4-1. Interleave Values

Controller	5 1/4-inch Peripherals	8-inch Peripherals
iSBC 214	Interleave is 1	--
iSBC 215G	Interleave is 4	Interleave is 5
iSBC 186/224A	Interleave is 1	--
SCSI Interface	Interleave is 4	--

Table 4-2. Physical Device Names

Device Name	MDP* Option Codes	System 310 * Designation	System 310AP * Option Codes	System 320 * Option Codes	Hard Disk Model Number
cm0 cm1		10 M-byte 5 1/4-inch Peripheral	- none -	- none - - none -	CMI 5412
cmb0 cmb1		17 M-byte 5 1/4-inch Peripheral	GK9 or GK10	- none - - none -	CMI 5419 or Fuji M2235
qma0 qma1		42 M-byte 5 1/4-inch Peripheral	GK2	T1GF2 T1GF7	Quantum Q540
mma0 mma1		140 M-byte 5 1/4-inch Peripheral	GK3	T1GF4 T1GF9	Maxtor XT-1140
mmb0 mmb1		86 M-byte 5 1/4-inch Peripheral	GK8	- none - - none -	Maxtor XT-1085
mmc0 mmc1		- none - - none -	- none -	T1GF10 T1GF12	Maxtor EXT-4175
mmd0 mmd1		- none - - none -	- none -	T1GF11 T1GF13	Maxtor EXT-4380
sma0 sma1		43 M-byte 5 1/4-inch Peripheral	GK12	T1GF14 T1GF15	Seagate SR251
tma0 tma1	T2GF3 T2GF8	85 M-byte 5 1/4-inch Peripheral	GK5	T1GF3 T1GF8	Toshiba MK56FB

* Refer to the Option List for the hard disk option codes. The Option List is supplied as either a separate sheet of paper in your kit and/or permanently fixed to the bottom of your system.

Table 4-3. Number of Files

Device Name	Number of Files
cm0 and cm1	1250
cmb0 and cmb1	2125
qma0 and qma1	5000
mma0 and mma1	17500
mmb0 and mmb1	10750
mmc0 and mmc1	21875
mmd0 and mmd1	47500
sma0 and sma1	6375
tma0 and tma1	10625

iRMX II supports other hard disk drives. Refer to the section on ATTACHDEVICE in the *Operator's Guide to the Extended iRMX II Human Interface* for the names of other drives.

All device names reflect the unit number of the drive. All device names that end with a "0" designate unit zero of the device; all device names that end with a "1" designate unit one of the device.

Examples of Option A, Formatting the Entire Hard Disk

A valid command to use for a microcomputer containing a Quantum Model Q540 hard disk configured as unit 0 controlled by the iSBC 186/224A or the iSBC 214 board would be:

```
SUBMIT :DISKETTE:FORM_DISK( QMA0, 1, 5000 ) <CR>
```

A valid command to use for a microcomputer containing a Maxtor Model XT-1140 hard disk configured as unit 0 controlled by the iSBC 186/224A or the iSBC 214 board would be:

```
SUBMIT :DISKETTE:FORM_DISK( MMA0, 1, 17500 ) <CR>
```

A valid command to use for a microcomputer containing a Toshiba Model MK56FB hard disk configured as unit 1 controlled by the iSBC 215G board would be:

```
SUBMIT :DISKETTE:FORM_DISK(TMA1, 4, 10625)
```

After the formatting of the hard disk is complete, the FORMAT command will issue a summary of the alternate tracks it assigned on the hard disk. Most hard disks contain bad tracks when they are delivered from the manufacturer so this is a normal condition, and assigning alternates is the method iRMX II uses to compensate for bad tracks. The message printed starts with the line:

```
The following tracks have been assigned an alternate:
```

.
.
{disk information}
.
.
.

Next, the DISKVERIFY command is executed to check the formatting of the hard disk. Following the DISKVERIFY command, the FORM_DISK.CSD SUBMIT file contains a series of the character 'y'. The 'y' is placed in the SUBMIT file to respond to any queries the DISKVERIFY command may issue. If the DISKVERIFY command does not make any queries, the error message 'illegal command' is issued which has no effect on the installation process. You should ignore this error message.

OPTION B: Formatting Only Track 0 of Your Hard Drive

This option formats only track zero of your system's hard disk and then places the second stage of the Bootstrap Loader on this track. No files on the disk will be affected by this option. You should choose this option if your disk was formatted using iRMX 286 Release 2.0 or iRMX 86 Release 7.0.

Note that you can also boot load the iRMX 286 Release 2.0 or iRMX 86 Release 7.0 Operating System from the same hard disk using the new second stage.

In selecting this option, you obtain the capabilities of the Bootstrap Loader's DEBUG switch. Refer to the *Extended iRMX II Bootstrap Loader Reference Manual* for details on the DEBUG switch.

To format only track 0 of your hard disk, type the SUBMIT command below:

where:

device name — is the physical name of your microcomputer's hard disk. Refer to Table 4-2 for the appropriate physical device names for hard disks used in Intel microcomputers. Note that the device name is not a logical name so it does not have colons surrounding it.

This SUBMIT file will ensure the system manager has full access to the directories and files affected by the installation process. It will also rename the following iRMX 286 R2.0 directories and then will re-create them for Release 3 installation:

Release 2.0 Name	Renamed to
:SD:SYS286	:SD:SYS286_R2
:SD:RMX286	:SD:RMX286_R2
:SD:USER	:SD:USER_R2
:SD:BOOT	:SD:BOOT_R2

This is done as a precautionary measure to avoid inadvertantly modifying any of your existing files. A later section describes how to merge the contents of the renamed directories with the newly created directories.

Performing this option will cause the DISKVERIFY command to be invoked using the "FIX" option. The "FIX" option automatically upgrades the file structure by including accurate checksums for each file.

As the DISKVERIFY utility executes, it will correct discrepancies it finds on the hard disk. Refer to the DISK VERIFICATION UTILITY USER'S GUIDE for information on the meaning of these messages. If your hard disk was previously formatted using the iRMX 86 Release 7 or iRMX 286 Release 1 Operating System, this SUBMIT file will compute checksum values for the file structure on the disk. When this occurs, the following message, indicating that corrective action has been taken, will be repeatedly printed to the screen.

```
FILE=(file name, fnode):LEVEL=level:PARENT=parent:TYPE=type
Bad Checksum : value, Should Be : checksum ... FIXED
```

Following the DISKVERIFY command, the FORM_TRK0.CSD SUBMIT file contains a series of the character 'y'. The 'y' is placed in the SUBMIT file to respond to any queries the DISKVERIFY command may issue. If the DISKVERIFY command does not make any queries, the error message 'illegal command' is issued which has no effect on the installation process. You should ignore this error message.

This option saves the old version and temporarily creates new versions of the system configuration files :CONFIG:TERMCAP, :CONFIG:TERMINALS, :CONFIG:UDF and all the :CONFIG:USER/<user_name> user configuration files.

Your old iRMX files now exist in the /RMX286_R2 directory. You may wish to generate other Release 2 versions of the Operating System. Refer to Retaining An Older Version of the Operating System for instructions on how to do this.

STEP 5: Diskette Installation of The Directory Structure

This step creates the standard directory structure on your hard disk. This step is required regardless of the the option chosen for formatting the hard disk.

The SUBMIT file will copy files from the Start-up System Commands Diskette **over** the corresponding files, if any exist, on your hard disk. The only way that this is possible is if you did not select any option from Step 4. If this is not acceptable, go back to Step 1. This SUBMIT file will also automatically cause the hard disk to become the system device for the iRMX II system.

To create the iRMX II directory structure and copy the Start-up System commands to your hard disk, enter the following command:

Where:

device name	is the physical name of your hard disk you used in the previous step. Refer to Table 4-2 for the appropriate physical device names for hard disks used in Intel microcomputers. Note that the device name is not a logical name so it does not have colons surrounding it.

As the SUBMIT command creates the iRMX II directory structure, a series of messages appear. If the system encounters an error during the process, it displays an error message but does not stop; the system continues executing the SUBMIT command until it reaches the end of the command. In the case where the directory already exists (as a result of choosing Option B in the previous step), or no file exists yet in the directory, the SUBMIT file will generate the warning messages:

```
<directory name>, file already exists
<file name>, file does not exist
```

These error messages are expected and can be ignored.

When the system displays any other error message, stop the system by typing a CONTROL-C and correct the fault. For example, errors can be caused by permission rights not being assigned correctly. After entering the CONTROL-C, detach the hard disk drive by typing: DD :W:<CR>, correct the problem, and enter the SUBMIT command again.

The last command in this SUBMIT file switches the system device from being the diskette drive to be the hard disk and restarts the system. The system device is the device from which the Operating System reads its commands. When you invoked the SUBMIT file "INSTAL_RMX.CSD", it was read from the Start-up System Commands diskette. Now the system will re-initialize and future commands will be read from the hard disk. You must again logon to the system specifying the name "super" and the password "passme" when prompted for them. You will be prompted to reset the date and time again. In any system that has a Global Clock (as on the iSBC 546 or MULTIBUS II iCSM boards) the date and time have not been set yet.

To set the DATE in the Global Clock, type:

```
date <date> global <CR>
```

where:

<date> has the form described in Step 3.

To set the TIME in the Global Clock, type:

```
time <time> global <CR>
```

where:

<time> has the form described in Step 3.

STEP 6: Installing The Extended iRMX® II Files

During this step you will use the diskettes from the iRMX II package. You will install the contents of the remaining Release Diskettes **except diskette number 20 which is for SERIES-IV systems only.**

Note that the device name you specify in this command is different from the device name you used to format the hard disk. The SUBMIT file will attach and detach the diskette drive for you automatically. All the remaining diskettes can be installed by repeating this single SUBMIT command. Remove Release Diskette Number 5 and insert Release Diskette Number 6. Enter the following command:

The command "INSTALL" is the alias for the HI command invocation "SUBMIT :CONFIG:CMD/INSTAL(WMF0)".

When the SUBMIT command finishes successfully, remove the diskette and insert Release Diskette Number 7 and enter the INSTALL command again. Repeat this step for each Release Diskette up through Release Diskette Number 19. You will have installed Release Diskettes Number 6 through 19 in this manner. Release Diskette Number 20 is used on Series-IV systems only.

The parameter "WMF0" is the physical device name of the diskette drive and is appropriate for the System 310 and 320 microcomputers. Use "WMF0" on MDP systems using the iSBC 186/224A controller. If you are installing from a diskette that is not named "WMF0", you must issue the full command invocation, specifying the correct device name for the diskette drive. Use "SMF0" on systems using the SCSI interface.

The :CONFIG:CMD/INSTAL.CSD file attaches and detaches the diskette drive for you automatically. It is commonly used to install the contents of Release Diskettes onto the iRMX II system. It requires that the diskette whose contents are being installed contains a SUBMIT file named "INSTAL.CSD".

As the SUBMIT command copies the iRMX II files to the hard disk, a series of messages appear. If the system encounters an error during the process, it displays an error message but does not stop; the system will continue executing the SUBMIT command until it reaches the end of the command.

When the system displays an error message, stop the system by typing a CONTROL-C, detach the diskette drive by typing: DD :F:<CR> and correct the fault. For example, errors can be caused by inserting the diskette incorrectly. After entering the CONTROL-C, detach the diskette drive, remove the diskette, reinsert the diskette correctly, and enter the SUBMIT command again.

Also, if you are not logged on as the system manager, access rights to files may cause errors. After entering the CONTROL-C, detach the diskette drive and logoff. Then logon as the system manager, correct the access rights and enter the SUBMIT command again while logged on as the system manager. Remember, you should always install files on your iRMX II system while logged on as the system manager.

STEP 7: Copying The Boot System Onto The Hard Disk

In this step you will install a bootable version of the Operating System on your hard disk. The SUBMIT file used in this step will attach and detach the diskette drive for you automatically. Remove Release Diskette Number 19 and re-insert Diskette Number 1, 2, 3, or 4. Use the same diskette that you used to boot the system in Step 3. Enter the following command:

STEP 8: Installing The Language Utilities

The next step is to install the iRMX II Language Utilities, supplied in a separate box from the rest of the iRMX II Operating System. Perform this step whether you installed the Operating System from tape or diskettes.

Before beginning the installation of the language utilities, check that you have these diskettes:

1. iAPX286 BINDER AND LIBRARIAN FOR iRMX II-BASED SYSTEMS 1 OF 5
2. iAPX286 MAPPER AND OVERLAY GENERATOR FOR iRMX II-BASED SYSTEMS 2 OF 5
3. iAPX286 SYSTEM BUILDER FOR iRMX II-BASED SYSTEMS 3 OF 5
4. iAPX286 MACRO ASSEMBLER FOR iRMX II-BASED SYSTEMS 4 OF 5
5. 80287 SUPPORT LIBRARY FOR iRMX II-BASED SYSTEMS 5 OF 5
6. PL/M-286 COMPILER FOR iRMX II-BASED SYSTEMS
7. iAPX86 UTILITIES PACKAGE FOR iRMX II-BASED SYSTEMS 1/2
8. iAPX86 UTILITIES PACKAGE FOR iRMX II-BASED SYSTEMS 2/2
9. iAPX86 MACRO ASSEMBLER PACKAGE FOR iRMX II-BASED SYSTEMS
10. PL/M-86 COMPILER FOR iRMX II-BASED SYSTEMS
11. iRMX II AEDIT Text Editor

To copy the iRMX II Language Utilities, insert each diskette into the diskette drive and enter the following command:

Where:

device name	is the physical name of the diskette drive. The parameter "WMF0" is the physical device name of the diskette drive and is appropriate for the System 310 and 320 microcomputers. Use "WMF0" on MDP systems using the iSBC 186/224A controller. If you are installing from a diskette that is not named "WMF0", you must issue the full command invocation, specifying the correct device name for the diskette drive. Use "SMF0" on systems using the SCSI interface.

This SUBMIT file will attach and detach the diskette drive for you automatically.

Again, if the system displays an error message, stop the system by typing a CONTROL-C, detach the diskette drive by typing "DD :F:" and correct the fault. Then re-invoke the SUBMIT file.

Once you have installed all the language diskettes, enter the following command:

This SUBMIT file copies the various libraries supplied with the languages you have just installed to the appropriate directories.

INSTALLING THE iC-286 COMPILER

To install the iC-286 compiler, Intel provides two SUBMIT files. To install the contents of the first iC-286 diskette, insert disk 1/2 and type:

To install the contents of the second iC-286 diskette, insert disk 2/2 and type:

INSTALLING THE FORTRAN-286 COMPILER

To install the Fortran-286 compiler, Intel provides two SUBMIT files. To install the contents of the first Fortran-286 diskette, insert disk 1/2 and type:

To install the contents of the second Fortran-286 diskette, insert disk 2/2 and type:

INSTALLING THE PASCAL-286 COMPILER

To install the Pascal-286 compiler, Intel provides three SUBMIT files. To install the contents of the first Pascal-286 diskette, insert disk 1/3 and type:

To install the contents of the second Pascal-286 diskette, insert disk 2/3 and type:

To install the contents of the third Pascal-286 diskette, insert disk 3/3 and type:

STEP 9. Preparing The Aedit Editor For Use With Your System

If you use Intel's Aedit text editor, your system must contain a macro file named AEDIT.MAC. This macro file must be located in either the directory :LANG: (logical name for the directory which has the pathname /LANG286) or the directory :HOME:. The purpose of this macro is to define the attributes of your terminal to Aedit. Intel provides a number of macro files that at this point have been installed in the :LANG: directory. These macro files are named for the type of terminal that they define (For example, VT100.mac defines the VT100 terminal.)

Enter:

The system displays the files located in the :LANG: directory with the .MAC extension. Locate one that matches the type of your terminal. If you are not sure what terminal a macro defines, you can use the COPY command to print the macro on the system console. Each macro starts with a comment giving the full name of the terminal.

Once you have located the macro that defines your terminal, assuming you are still in SUPER mode from the previous step, type:

When this command completes, you can use Aedit as the user "world". You should always be the system manager when you manipulate system files.

In cases where there is no macro that defines your terminal, first check to see if your terminal can emulate one of the terminals for which a macro is supplied. If this fails, you must write your own macro. Refer to the Aedit manual included in the languages package for information on writing macros.

If you are installing the iRMX II Operating System on a microcomputer which will host multiple users (will be multi-user), you can copy the appropriate Aedit macro file into each user's :HOME: directory. This will allow the Aedit editor to be invoked by users who have different types of terminals. In this case, there should be no copy of AEDIT.MAC in the :LANG: directory.

STEP 10: Installing iRMX® Networking Software

Intel provides the iRMX Networking Software product (iRMX-NET) for use in building Ethernet-based Local Area Networks (LANs). iRMX-NET allows extended iRMX II systems to share files with other extended iRMX II systems as well as iRMX 86, Xenix, iNDX, MS-DOS and VAX/VMS systems. If you are not installing the iRMX-NET Software, proceed to the next step.

If you have previously installed iRMX-NET and you chose the "Format Track Zero" option of Step 4 your iRMX-NET files have been saved. They now reside in a different directory than prior to this installation. This is a normal function of the "Format Track Zero" option of Step 4. You must install the iRMX II Update and execute the RMXMRG.CSD SUBMIT file described in the next step.

The information beyond this point is for users who are installing iRMX-NET for the first time or are installing a new release of iRMX-NET.

To install the iRMX-NET Networking Software, refer to the iRMX-NET Networking Software User's Guide which contains a description of the product and how to install it on System 300 Series Microcomputers. You can use the alias command 'INSTALL' to install the iRMX-NET files. After completing the iRMX-NET installation, return to the next step in this manual. Do not try to generate a version of the extended iRMX II Operating System which includes the iRMX-NET software at this time.

Note that the definition files supplied with iRMX-NET are very similar to those supplied with the iRMX II Operating System. But since they are released at different times, the iRMX-NET definition files may indicate that they do not match the version of the ICU when the ICU is invoked. This is normal,and you must use the ICU Restore feature to upgrade the definition files. (This Restore feature is not the HI RESTORE command.) Refer to the *Guide To The Extended iRMX II Interactive Configuration Utility* for details on the Restore feature of the ICU.

STEP 11. Installing The Update Package

An important phase of installing the iRMX II Operating System is the installation of the current iRMX II Update Package. You must perform this step even if you are installing a **new** release of the operating system. The update package is Intel's mechanism for fixing any software problems identified in the current version of the software. If you do not apply the update, you will be working with an outdated version of the iRMX II Operating System.

The Update Package accompanies all the shipments of the iRMX II Operating System. (The Update Package is shipped in a separate box.) Each Update consists of one or more Update Diskettes, an Update Installation Guide, Update change pages to the manual set and a customer letter.

The Update Diskette contains all of the fixes (ZAPs) to be applied to the iRMX II Operating System. The diskette is labeled:

"RMXII UP**x**R**y.z** "

x is the release level of the Update Package

y.z is the release level of the Operating System

The Update Installation Guide contains both detailed descriptions of each fix (ZAP) and detailed instructions on how to install the Update Package.

To install the Update to your system, find the Update Package, which is shipped in a separate box from the iRMX II Operating System, and follow the instructions in the Update Installation Guide. Also, make certain to read the Update Package Customer Letter for additional information on the Update Package.

STEP 12: Combining Directories when Upgrading to Release 3.0

If you invoked the FORM_DISK.CSD SUBMIT file in Step 4 (you formatted your entire hard disk), this step does not apply to you. Proceed to the next step.

Upgrading From iRMX® 286 Release 2.0

The following command combines the required files from iRMX 286 Release 2.0 into this release of the iRMX II Operating System. To perform this step, type the following command:

Upgrading From iRMX® 86 Release 7.0

If you are upgrading to iRMX II from iRMX 86 Release 7.0, type the following command to combine the required files from iRMX 86 directories into the iRMX II directory structure:

If you are upgrading to iRMX II from an earlier iRMX release, there are no files to merge and you should proceed to the next step.

STEP 13: Generating An Updated Version Of The Operating System

Now that you have installed the latest Update, you will generate an updated version of the Operating System. The version of the Operating System you loaded from the Start-up System Boot Diskette was generated from the original iRMX II libraries and, as such, does not contain any enhancements or corrections to problems. (Note that the version you generate in this step contains the System Debugger, the Start-up System does not).

Logoff from the system by typing:

```
-
```

The logon banner and prompt will be displayed. Now log back on to the system using the name "world" and a carriage-return for the password. It is not necessary to be the system manager to generate a configuration of the Operating System. You will be prompted to reset the date and time again. If your system contains a Global Clock, the date and time are still correct. Respond with "e" (EXIT) to leave the date and time at their current values.

Select an ICU definition file from those provided with iRMX II. Choose the default definition file which matches your microcomputer from those listed below.

ICU Definition File	CPU Boards Supported
28612.DEF	iSBC 286/10(A), iSBC 286/12
38620.DEF	iSBC 386/2x, iSBC 386/3x
286100A.DEF	iSBC 286/100A
386100.DEF	iSBC 386/116, iSBC 386/120

To generate the iRMX II Operating System, you will select the example commands that match your system. These examples follow the explanation below.

```
-
-
-
```

<boot_file_name> is the name of an ICU definition file **without the .DEF extension**. The ALIAS "CRDIR" is created for the HI command ":SYSTEM:CREATEDIR". Typing this alias will create a directory in which to generate an Updated system. The system generated is specified by the ICU definition file you select. The command "AF" is the alias for the HI command ":SYSTEM:ATTACHFILE". Typing these commands will create a directory for the system generation underneath your :HOME: directory and attach it as your current default directory.

The command "MKSYS" is the alias for the HI command invocation SUBMIT :CONFIG:CMD/MKSYS(#0). The #0 parameter in the MKSYS alias command receives the "boot_file_name". This convention of using the same name for the generation directory, definition file and created system is helpful if you generate several systems during the development of your application.

The Bootstrap Loader requires both a third stage and the Operating System itself in order to bootstrap load. The MKSYS command copies a third stage, which has a name that matches the "boot_file_name" name used in the MKSYS invocation, from the directory /RMX286/BOOT to the directory /BOOT. The installation process places third stages with the following names in the directory /RMX286/BOOT: 28612, 38620, 286100A, SXM386, and 386100.

If you invoked the FORM_TRK0.CSD SUBMIT file in Step 4, you will probably have a directory in your :HOME: directory with a name identical to the one you will create next. Rename this directory now by typing:

An example of doing this, assuming that your system contains an iSBC 28612 processor board is:

```
RENAME 28612 to 28612_r2
```

If your processor board is the iSBC 286/10(A) or the iSBC 286/12 in a System 286/310 Microcomputer, select the 28612.DEF definition file and type:

```
-CRDIR 28612 <CR>
-AF 28612 <CR>
-MKSYS 28612 <CR>
```

This will create a new version of the iRMX II System named "/BOOT/28612.286". The MKSYS command will also make a copy of the Bootstrap Loader third stage appropriate for the new version of the Operating System named "/BOOT/28612".

If your processor board is the iSBC 386/2x or iSBC 386/3x, select the 38620.DEF definition file and type:

```
-CRDIR 38620 <CR>
-AF 38620 <CR>
-MKSYS 38620 <CR>
```

This will create a new version of the iRMX II System named "/BOOT/38620.286". The MKSYS command will also make a copy of the Bootstrap Loader third stage appropriate for the new version of the Operating System named "/BOOT/38620".

If your processor board is the iSBC 286/100A, select the 286100A.DEF definition file and type:

```
-CRDIR 286100A <CR>
-AF 286100A <CR>
-MKSYS 286100A <CR>
```

This will create a new version of the iRMX II System named "/BOOT/286100A.286". The MKSYS command will also make a copy of the Bootstrap Loader third stage appropriate for the new version of the Operating System named "/BOOT/286100A".

If your processor board is the iSBC 386/100/116/120, type:

```
-CRDIR 386100 <CR>
-AF 386100 <CR>
-MKSYS 386100 <CR>
```

This will create a new version of the System named "/BOOT/386100.286". The MKSYS command will also make a copy of the Bootstrap Loader third stage appropriate for the new version of the Operating System named "/BOOT/386100".

Execution of MKSYS may cause the main screen of the ICU to scroll several times. This is expected and does not indicate a problem.

After the MKSYS command completes, check the file, <boot_file_name>.out, for any errors that may have occurred during system generation. There are two methods of doing this check, you can use AEDIT if you installed it, or you can use the Human Interface COPY command. To use the COPY command, type:

```
-
```

You can use CONTROL-W to page through the file; each time you enter CONTROL-W your display will scroll one screen. Entering a CONTROL-Q exits this scrolling mode. CONTROL-S can also be used to halt the file during printing to the terminal, CONTROL-Q resumes printing to the terminal. Any errors indicate a problem. A complete listing of a generation output file is shown in the *Guide To The iRMX II Interactive Configuration Utility*.

Do not invoke the MKSYS command after you have successfully used it once to generate the Operating System. Instead, use the local copy of the definition file created for you by the MKSYS command. This version of the definition file can be the starting point to develop your custom application system.

You can create an ALIAS that makes invoking the ICU easier.To do this, use your editor to add the following line to your :PROG:ALIAS.CSD file:

To initialize the new ALIAS, type:

After doing this you can invoke the ICU by typing:

```
icu <pathname>/<definition file>.DEF <CR>
```

If you do not add this ALIAS, when you invoke the ICU, type:

```
/RMX286/ICU/ICU286  <pathname>/<definition file>.DEF <CR>
```

You can also use an ICU definition file that you created in Release 2.0 of the iRMX 286 Operating System with the Release 3.0 ICU.

STEP 14: Booting The Operating System From A Hard Disk

After the system has executed the SUBMIT command described in Step 13, the hard disk contains a version of the iRMX II Operating System suited to your CPU board. To test this version of the Operating System, you must shutdown the system properly by invoking the SHUTDOWN command. To do this, invoke the SUPER command and enter the password 'passme' when prompted. This causes you to become the system manager without logging on as the user "super". Type:

```
-
```

Respond with the password 'passme' to the password prompt. Now, type:

```
-
```

The command "SH" is the alias for the HI command invocation ":SYSTEM:SHUTDOWN W=0". After the SHUTDOWN command displays its shutdown complete message, reset the system by pressing the RESET button or turning the RESET switch on the front panel of your microcomputer.

The SCT, described earlier, starts executing, enter an uppercase U in response to the x's being printed on the screen. When your system console displays the monitor prompt, enter one of the following monitor commands to bootstrap load the newly created version of the Operating System:

```
.                                   (iSDM monitor)

>                                     (DMON-386 monitor)
```

where:

<definition_name>	is the same name you used in Step 13 which reflects the particular CPU board in your microcomputer.

If you used the definition file, 28612.DEF, type: b /boot/28612

If you used the definition file 38620.DEF type: b /boot/38620

If you used the definition file SXM386.DEF type: b /boot/sxm386

If you used the definition file, 286100A.DEF, type: b /boot/286100A.

If you used the definition file, 386100.DEF, type: b /boot/386100.

Once the Operating System bootstrap loads and signs on, it indicates you have successfully generated a version of the Operating System. The logon banner and prompt will be displayed. Now log back on to the system using the name "world" and a carriage-return for the password. You will also be asked to set the date and time again. If you do not wish to do this again, respond with "e" when asked to enter the valucs.

Once you are satisfied with the new system, copy the new version of the system over the previous version. To do this, you must invoke the SUPER command and enter the password 'passme'. Now, type:

-

where:

<boot_file_name>	is the same name you used in Step 13 to generate the Operating System.

If you do not specify a path name when entering the iSDM or DMON boot command, the Bootstrap Loader will load the default boot system. The pathname /SYSTEM/RMX86 is the default pathname for the Bootstrap Loader.

Note that a valid Bootstrap Loader third stage is provided in this step. When you create your own systems, you must remember to copy the correct third stage to the /BOOT path.

For a detailed description of the standard definition files, refer to Chapter 6.

STEP 15: Retaining an Older Version of the Operating System

This step is intended for users who chose the "Format Track Zero" option of Step 4. It explains how to move your renamed files to the directories where you want them. The following paragraphs explain the effects of the installation process on a iRMX II Release 2 system and an iRMX I Release 7.0 system.

An iRMX II Release 1.0 directory structure on the hard disk will be affected by the Release 3.0 installation process in these ways:

- The directory /USER will be re-created and will contain the logon file "R?LOGON" which is specific for iRMX II.3. It will execute a SUBMIT command named "ALIAS.CSD" which is not valid for an iRMX 286 R1.0 system but will have no adverse action if executed. If this is not acceptable, you should change the user's configuration file in the Release 1.0 configuration directory /CONFIG/USER/<user ID> to select the user's Release 1.0 renamed home directory. Later you can combine the files in the home directories.

An iRMX II Release 2.0 directory structure on the hard disk will be affected by the Release 3.0 installation process in these ways:

- The installation process renames the contents of the iRMX II Release 2.0 directories affected by this installation. The directories affected are: :SD:SYS286, :SD:RMX286 and :SD:BOOT. The FORM_TRK0.CSD SUBMIT file executed in Step 4 automatically renames them for you. The following names are the names used by the SUBMIT file:

Release 2.0 Name	Renamed to
:SD:SYS286	:SD:SYS286_R2
:SD:RMX286	:SD:RMX286_R2
:SD:BOOT	:SD:BOOT_R2

An iRMX 86 directory structure on the volume will be affected by this installation process in three ways:

1. The default boot system will be changed. This file has the pathname "/SYSTEM/RMX86". You must either rename the default boot system (e.g. /system/rmx86.86), make a copy of the boot system (e.g. copy /system/rmx86 to /boot86/rmx86) which you can use for booting, or you must rename the SYSTEM directory.
2. The directory /USER will be renamed to /USER_R2 and a new /USER directory will be created.
3. As an effect of change 2, the logon file "R?LOGON" will be specific for iRMX II. It will execute a SUBMIT command named "ALIAS.CSD" which is not valid for

CHAPTER 2
USING THE HUMAN INTERFACE

2.1 INTRODUCTION

This chapter discusses how to use the Human Interface. It doesn't provide detailed descriptions of individual commands; these are in Chapters 3 and 4. However, it does address the following topics:

- Requirements for including the Human Interface in your system.
- Configurable features of the Human Interface.
- The process of loading and accessing the Human Interface, including static and dynamic logon.
- The iRMX II file structure and file-naming conventions (including wild cards).
- Devices supported by the Human Interface.
- Automatic Device Recognition.
- The general syntax of a command.
- The system manager.
- Adding non-resident users to the system

2.2 SOFTWARE REQUIREMENTS

The Human Interface is a layer of the iRMX II Operating System. Therefore, it requires the inclusion of other layers to make it a part of your application system. To include the Human Interface in your application system, you must also include

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

During command execution, the Human Interface invokes the services of these layers in a way that is transparent to the operator. Therefore, an operator needs little knowledge of operating system structures to load and execute programs from the console keyboard.

2.3 CONFIGURABLE FEATURES OF THE HUMAN INTERFACE

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

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

  If your Human Interface is configured for single-user support (you are the only user accessing the system.), you are less interested in this information. You have no great concern about file access rights since all the files on the system are yours. You may wish to deny yourself delete access to prevent accidental deletion of important files.

  This manual contains information for both types of users. It explains all the information needed for a multi-user system, but it also points out cases where information does not apply to single-user systems. In all cases, the information for single-user systems is a subset of the information for multi-user systems.

- **Dynamic and Static Logon**: In a multi-user environment, the Human Interface uses a logon procedure to ensure that operators are valid users. You can configure any Human Interface terminal for one of two kinds of logon: dynamic or static.

  With dynamic logon, the Human Interface assumes that different operators will be using the same terminal at different times. Therefore, it prompts operators to enter a logon name and a password before granting them access to the system. If you want to use the networking capabilities provided by iRMX Networking Software, you must use dynamic logon.

  With static logon, the Human Interface assumes that only a single operator uses a particular terminal. Therefore, the Human Interface knows which logon name to use and does not require the operator to enter that information.

  This manual describes both static and dynamic logon, and how to add non-resident users (users that need to log in) to the system.

- **Initial program**: Immediately after logon, the Human Interface starts an initial program for each terminal. This initial program is usually a Command Line Interpreter (CLI), a program that reads and executes commands.

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

Other configuration options also affect how the system appears to a user. If you are not involved in iRMX II configuration, contact your system manager for more information.

2.4 LOADING THE OPERATING SYSTEM

Before you can access the Human Interface, the operating system must be loaded into memory and started. (If you have previously burned your system into PROM, it will be loaded automatically at power on.) This process can vary from system to system (depending on such things as the monitor you use), but generally it involves one of these procedures:

- Using the iRMX Bootstrap Loader to load the operating system from iRMX II secondary storage to memory. This can be done during development or after the system is completed. The Bootstrap Loader will load a system on both an iSBC 386/2X-based computer (MULTIBUS I system) and an iSBC 386/1xx-based computer (MULTIBUS II system) in about one minute.
- Connecting the target system (the iRMX II system) to an Intel Microcomputer Development System and using the iSDM package to load the operating system from the host development system to memory in the iRMX II target system. This procedure is normally done during the development phase of an application system, when some of the system elements are still undergoing development. Refer to the *iSDM System Debug Monitor Reference Manual* for more information. Note that this method cannot be used if you are using the D-MON386 monitor The D-MON386 monitor resides only on a target system.

2.4.1 Loading the Operating System on an OEM System

If your system is not one of Intel's family of integrated systems you should perform these steps:

1. Reset the system; usually, reset involves pressing a RESET button on the system chassis. A series of characters (usually lowercase x's) should appear at 9600 baud on the system terminal (the one connected to the processor board).

2. Type a series of uppercase U's at the system terminal; this allows the resident monitor to determine the system terminal's baud rate. The monitor displays the following information:

```
<monitor> Vx.y
COPYRIGHT <year> Intel Corporation

.
```

The period (.) is the monitor prompt. The term <monitor> indicates which monitor you are using and Vx.y the version of the monitor.

3. Use the monitor's B command to bootstrap load the operating system. In most cases, you do this by entering

For the default configuration of the Bootstrap Loader, this command loads a file with pathname/SYSTEM/RMX86.286 from the first available boot device. It loads the command using a Bootstrap Loader whose third stage is located in /SYSTEM/RMX86. If your operating system resides in a file with a different pathname, you must specify the full pathname when you enter this command. An example of this is

Refer to the *Extended iRMX II Bootstrap Loader Reference Manual* for instructions on booting such a system.

2.4.2 Loading the Operating System on Intel Integrated Systems

On Intel System 300 Series microcomputers (such as the System 286/310), you should perform these steps:

1. Turn on the power to the terminals and to the system. If your system contains multiple chassis (such as the 286/380 Microcomputer System), turn on the power to the PROCESSOR chassis before turning on the power to the PERIPHERAL chassis.

2. Within a few seconds, the terminal connected to the system's processor board should display a series of lowercase x's. There is no user input required to load the operating system from your hard disk. In approximately 12 seconds, the System Confidence Test (SCT), a diagnostic program residing in PROM which performs an initial check of your hardware, begins to run automatically. The system assumes a baud rate of 9600 for your terminal.

3. When the SCT is successful, it invokes the Bootstrap Loader, which attempts to load a file with the pathname /SYSTEM/RMX86. The Bootstrap Loader and the SCT reside in PROM.

4. When the Bootstrap Loader completes loading, you can access the Human Interface from any terminal, as described in the next section.

2.4.3 Loading the Operating System Without a Resident Monitor

If your system does not include the monitor, simply ensure that an iRMX II-formatted volume containing the operating system resides on the proper device, and reset the system.

2.4.4 Loading a PROM-Based Operating System

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

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

2.5 ACCESSING THE HUMAN INTERFACE

Once the operating system software has been loaded, you access the Human Interface by logging on. If you're using a modem to communicate between your terminal and the iRMX II system, you must set up the modem link before you log on. The type of logon procedure you perform depends on the classification of your terminal. In an iRMX II-based system, there are two terminal classifications: static logon terminals and dynamic logon terminals.

Dynamic Logon Terminal:

- Dynamic logon terminals are not permanently associated with any user. A temporary association is formed whenever an operator logs onto the system. Therefore, a dynamic terminal can be associated with any number of user.
- To log onto a dynamic logon terminal, a user must enter a logon name and a password.
- Users should log off dynamic terminals when they finish their work. Once one user logs off a terminal, another user can use it.

Static Logon Terminal:

- Static logon terminals are permanently associated with a particular user. Whenever a static logon terminal is used, all the actions (creating files or invoking commands, for example) are performed under the same user name and ID, no matter which operator is actually using the terminal. This means that any number of operators can use the terminal, but the operating system recognizes them as a single user. The user associated with the static terminal is set during system configuration.
- With static logon terminals, users do not enter a name and password information to logon. Instead, the Human Interface handles the logon automatically. After powering up a static logon terminal and booting the operating system, a user must only perform the start up actions determined by the contents of the :PROG:R?LOGON file. Typically, this file contains commands that enable the user to set the date and time for the Human Interface, if there is no global clock in the system.
- Users cannot log off static logon terminals. Once the operating system associates a user with the static logon terminal, that terminal is set up with a permanent set of resources. The association between the static logon terminal and the user can be modified only by changing the :CONFIG:TERMINALS file and rebooting the application system.
- When a user stops using (logoffs) a static logon terminal, another user with the same attributes is initialized.

Special configuration files list information about all the terminals through which operators can communicate with the operating system. The information in these files specifies whether the terminals are static or dynamic logon terminals. Refer to the section titled "Adding Non-resident Users to Your System" at the end of this chapter for information on how to configure the terminals in your application system.

To log onto the operating system, either from a static logon terminal or a dynamic logon terminal, you must include other configuration files that list your logon name, your password, your user ID, and other information about you. Refer to the section titled "Adding Non-resident Users to Your System" at the end of this chapter for information about adding new users.

2.5.1 Modem Control

You may wish to use modems to communicate with the Operating System. This section explains how to set up a modem for use on the iRMX II end of a modem link. No information is given on how to attach or use the modem on the terminal end of the link.

Before attempting to use a modem with an iRMX II system, ensure that the iRMX II port you are connecting the modem to is configured as a modem link. Use the Interactive Configuration Utility (ICU) to set the "(MC) Modem Control" parameter of your terminal driver to "Yes" Refer to the *Extended iRMX II Device Driver's User' Guide* for more information.

2.5.1.1 Setting Up Your Modem

The following instructions detail how to set the switches for the Hayes-compatible smart modem.

SWITCHES	SETTING
DTR (Data Terminal Ready)	Normal
DSR (Data Set Ready)	Normal
Carrier Detect	Normal
Auto Answer	Enabled
Command Response	Suppress
Command Mode	Disabled

Once these switches have been set, and both sides of the modem link are properly cabled, your modem link is ready. The next section explains what to enter from the terminal to access iRMX II.

2.5.1.2 Establishing a Connection

Use the modem on the terminal end of the link to dial up the iRMX II system. When the modem on the iRMX II system answers, the Dynamic Logon cusp will assert the DTR signal. The word "CONNECT" will appear on your terminal. If the modem on the iRMX II end of the link has an auto-baud search, enter four capital "U's" (at a rate of about two per second) and a carriage return <CR>. If your terminal is not configured for auto-baud search enter a carriage return <CR>. Your terminal should now respond as if it were connected to the iRMX II system by a dedicated line. Note that the response from a modem link operates at the baud rate of the modem and therefore may be noticeably slower than a true dedicated line.

2.5.2 Accessing the Human Interface From a Static Logon Terminal

Assuming that the operating system is loaded, you can access the Human Interface through a static logon terminal simply by powering on the terminal. If the terminal is configured for automatic baud rate recognition, you must also enter the following character at the keyboard:

```
(uppercase U)
```

Continue to enter this character until it is echoed on the screen, then enter a carriage return

This character allows the operating system to determine the baud rate of your terminal.

When the Human Interface starts running, it logs you onto the system automatically and creates an environment for you to enter commands. This environment is an iRMX II job, which this manual refers to as an interactive job.

As part of the automatic logon process, the Human Interface assigns you a user ID. This user ID is your "identity" in the system. It determines your access to files and devices. Whenever you create files, the Human Interface assigns your user ID as the owner ID of the file. Being the owner of a file gives you complete control over the file; you can read it, delete it, write to it, update it, and grant access rights to other users. Thus, your ability to access files created by other users depends on the access they grant you.

Once the interactive job has been created by the operating system, an initial program begins execution. The initial program that runs in your interactive job (at your terminal) may be different from one that runs at another terminal. (A configuration option specifies which initial programs are associated with which user IDs.) Initial programs are generally Command Line Interpreters (CLIs), which read and parse command input and start programs running based on that input. The Human Interface supplies a standard CLI, which this manual assumes you are using.

The standard CLI begins by displaying the following header message and prompt:

```
iRMX II HI CLI, V<x.y>; USER = <user ID>
Copyright <years> Intel Corporation
<configurable sign-on message>
-
```

where

V<x.y>	The version number of the CLI.
USER = <user ID>	A display of your user ID. The Human Interface uses this ID to determine the type of access you have to files and devices. Most single-user systems are set up to give you an ID of WORLD (65535 decimal), but some may differ. Every multi-user user has read and write access to files created by WORLD. For this reason, multi-user systems should use the ID WORLD (65535 decimal) for files that will be accessed by multiple users.
<configurable sign-on message>	The contents of the file :CONFIG:SIGNON.MSG copied to the screen.

- (hyphen) The Human Interface default prompt. This prompt tells you the CLI is ready to accept command input. (You can change the prompt with the CLI SET command; see Chapter 3.)

If the information that appears at your terminal is different from this, contact your system manager to determine the differences between your initial program and the standard CLI.

Next, the standard CLI searches for the logon command file, a file whose pathname is :PROG:R?LOGON (later sections of this chapter discuss pathnames of files). A :PROG:R?LOGON file may exist for each user of the system and may include commands which assign the date or the minimum and maximum memory pool sizes for background jobs (see Chapter 3). The CLI expects to find command invocation lines in this file. When it finds this file, the CLI automatically invokes the SUBMIT command to process all the commands in the file (refer to Chapter 3 for more information about SUBMIT). You can modify the information in your :PROG:R?LOGON file to change the amount of processing that occurs automatically when the operating system recognizes your terminal. If the Human Interface does not find a R?LOGON file, it returns the message "file does not exist", and continues executing.

As supplied with the start-up versions of the operating system (on the Installation media), the R?LOGON file for each user contains the DATE and TIME commands which ask you for the correct date and time as follows:

```
-date q
<current date and time> <clock type>
DATE:
```

If the application system uses a global clock supported by the Basic I/O System, then the time and date are automatically initialized to the values in the clock.

In response, enter the date in any of these formats:

```
or
```

Entering just an "e" and a carriage return will retain the current date.If you use an improper format, the DATE discards your entry and prompts you for another date. For more information, refer to the description of the DATE command in Chapter 4.

After you enter the date correctly, DATE responds by displaying the date. You are then prompted for the time as follows:

```
-time q
<current date and time> <clock type>
TIME:
```

In response, enter the correct time in this format:

```
hours:minutes:seconds
```

You can omit the last field or the last two fields. TIME sets the omitted fields to zero. The following are all valid times:

```
or
```

Entering "e" followed by a carriage return retains the current time.

For more information, refer to the description of TIME in Chapter 4. TIME responds by displaying the date and time. After issuing "DATE" and "TIME", the default logon file submits ":PROG:ALIAS.CSD". After processing all the commands in the logon file, the CLI issues its prompt and returns control to you. You can then enter CLI or Human Interface commands, or invoke application programs.

2.5.3 Accessing the Human Interface From a Dynamic Logon Terminal

Assuming that the operating system is running, you can access the Human Interface through a dynamic logon terminal by powering on the terminal and supplying information about who you are. This dialogue with the Human Interface is called dynamic logon.

During dynamic logon, you supply a logon name and a password. Then the Human Interface scans the appropriate configuration files to ensure that you are a valid user. The configuration files also contain other information about you, such as your user ID, priority, and memory partition size. The Human Interface uses this information to create your interactive job--the environment in which you enter commands.

After logging on, any files you create are assigned your user ID. Your ability to access other users files depends on the access they grant to your user ID.

Dynamic logon is different than static logon because a terminal designated as a dynamic logon terminal is not permanently associated with one user ID. A terminal defined as static, however, is always associated with one user ID. Therefore, the operating system identifies any process started from a static terminal by the one user ID associated with that terminal, regardless of the operator who invoked the process. On the other hand, the operating system identifies a process started from a dynamic logon terminal by the ID of the person currently logged on.

Once someone logs off a dynamic logon terminal, other users with different user IDs can use it. As a result, an application system can accommodate many different kinds of users. The system manager can maintain security by assigning unique user IDs and by limiting access to files based on those IDs.

In addition to freeing a terminal for further use by other operators, logging off a dynamic logon terminal frees the memory pool given to the interactive job by the operating system. This freed memory pool can then be used by other jobs within the application system.

Before you attempt to perform dynamic logon, do the following:

- Be sure you are using a dynamic terminal. If you access the Human Interface from a static terminal, the files and programs you create will be associated with the static terminal's user ID, not the ID that you originally desired.
- Ensure that you have the correct logon name and password. If you don't know this information, consult your system manager.

Now you are ready to log onto the system. If your terminal is configured for automatic baud rate recognition, you must also enter the following character at the keyboard:

```
(uppercase U)
```

until the "U" is echoed on the screen, then enter **<CR>**.

This character allows the operating system to determine the baud rate of your terminal.

Once the baud rate has been determined, the following prompt appears:

```
<logon message>
Logon:
```

Where <logon message> is the contents of the file :CONFIG:LOGON.MSG.

Type the logon name assigned to you (followed by a carriage return). Once you have entered your logon name, the system prompts you for a password. If you have been assigned a logon name, but no password, press carriage return. Once you gain access to the Human Interface, you can give yourself a password by invoking the Human Interface command PASSWORD.

If you enter an incorrect logon name or password, the operating system keeps prompting you with the "logon" and "password" messages. There is no limit to the number of logon attempts that you can make, unless your terminal is connected to a modem. In this case, three unsuccessful attempts to log onto the system cause the modem to "drop" the Data Terminal Ready (DTR) line, which hangs up the phone. If this happens, you must dial the phone number again to relink to the remote iRMX II system.

After you enter a valid logon name and password, an initial program begins running. If there is not enough memory in the system to activate the initial program, the Human Interface will display a warning. In this case, consult your system manager.

The initial program that runs in your interactive job (at your terminal) might be different than one that runs at another terminal. This manual assumes that the initial program for all users is the Human Interface CLI, which displays the following (configurable) header message and prompt:

```
iRMX II HI CLI, V<x.y>: USER = <user ID>
Copyright <year> Intel Corporation
<configurable sign-on message>
-
```

where

V<x.y>	The version number of the CLI.
USER = <user ID>	A display of your user ID. The Human Interface uses this ID to determine the type of access you have to files and devices.
<configurable sign-on message>	A message that can vary from system to system. The text for this message is contained in the file :CONFIG:SIGNON.MSG. The Human Interface automatically displays this text whenever you log on.
- (hyphen)	The Human Interface default prompt. This prompt implies that the CLI is ready to accept command input. (You can change the prompt with the SET command; see Chapter 3.)

Once the header message is displayed, the system reacts the same way it does for input from a static terminal. This means, the CLI searches for the file :PROG:R?LOGON and invokes the commands in that file as explained in the previous section. After processing the commands, the CLI issues its prompt (-) and returns control to you. You can then enter CLI or Human Interface commands, or invoke application programs.

You end a session at dynamic logon terminal by typing LOGOFF. If your initial program is the standard CLI, the CLI searches for the file :PROG:R?LOGOFF and invokes all the commands in that file. As with the logon file, you can place any commands you wish into this logoff file, and they will be invoked whenever you log off.

If you are using a customized CLI, the LOGOFF command can be entered. However, it performs differently (see Chapter 4).

2.5.4 Relationship Between Dynamic Logon and Remote File Access

If your iRMX II system is configured to include the iRMX networking software, any user on your system who gains access to the Human Interface through dynamic logon automatically becomes a "verified user." In an OpenNET network system, a verified user can access files on remote systems through iRMX-NET. For more information about iRMX-NET, refer to the *iRMX Networking Software User's Guide*.

2.6 FILE STRUCTURE

One of the primary uses of Human Interface commands is to manipulate files. Before you use the Human Interface commands described in Chapter 4, you should have an understanding of the types of files that exist in an iRMX II environment and how to access those files.

2.6.1 Types of Files

The iRMX II environment has four types of files: named, physical, stream, and remote.

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

Physical files	Physical files are mechanisms by which the operating system accesses an entire I/O device as a single file. The Human Interface accesses backup volumes and devices such as line printers and terminals in this manner. It also accesses secondary storage devices (such as disk drives) as physical devices when formatting them. When operators access physical files, it is usually in a manner that is transparent to them (such as copying a named file to the line printer or formatting a disk).
Stream files	Stream files are mechanisms for communicating between programs. Two programs can use a stream file for communication if one program writes information to the stream file while another program reads the information. Terminal operators seldom use stream files directly.
Remote files	Remote files are the same as named files, except that they reside on a remote system connected to the OpenNET network using the iRMX Networking Software. No special semantics are needed to access remote files. For more information on remote files, see the *iRMX Networking Software User's Guide*.

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

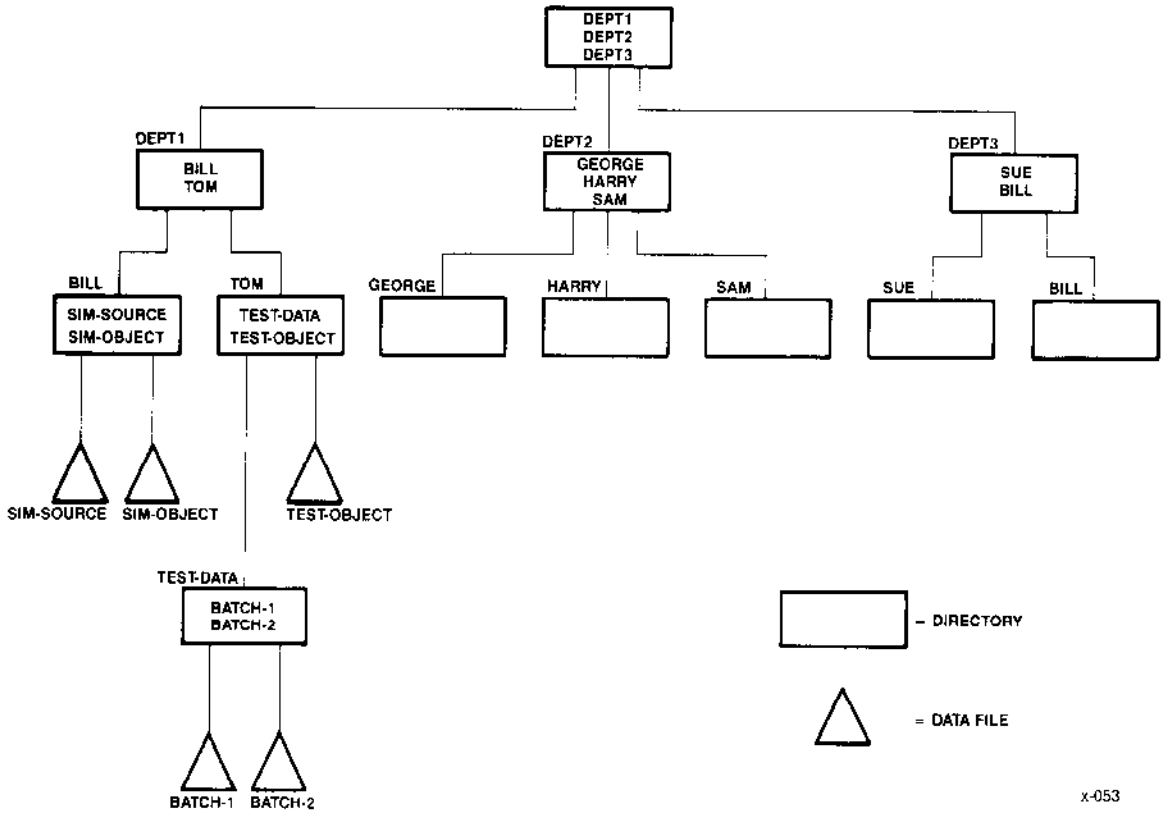


Figure 2-1. Example of a Named-File Tree

2.6.2 Named File Hierarchy

You can organize named files into structures called file trees, as shown in Figure 2-1. Figure 2-2 (parts 1 through 4) shows the actual default file structure that the installation process builds on your system. Intel recommends that you retain the default file structure because the update service process assumes this structure.

As you can see from the figure, the file tree has two kinds of files: data files and directories. Data files (shown as triangles in Figure 2-1) contain the information that you manipulate during your terminal session (for example, inventory, accounts payable, text, source code, and object code). Directories (shown as rectangles in Figure 2-1) contain only pointers to other files (either named files or directories). You can have multiple directories in a hierarchical structure so that instead of having a single directory containing an enormous number of files, you can organize your files into logical groups under several directories. You can display the list of files in any directory by invoking the DIR command for that directory (refer to Chapter 4 for more information).

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

Each file tree resides on a secondary storage volume--the storage medium that contains the data. Examples of volumes include flexible disks, hard disks, RAM disks, and bubble memories. Before you can place named files on a volume, you must format the volume to accept named files. The formatting process writes a number of data structures on the volume to aid the operating system in creating and maintaining files. You can use the FORMAT command (described in Chapter 4) to format your volumes.

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

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

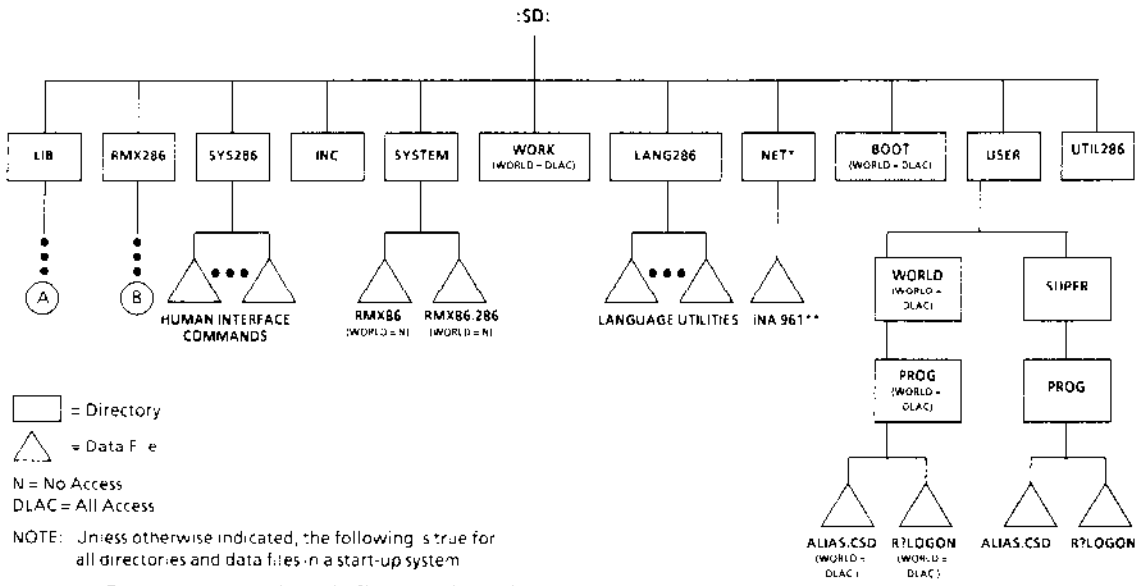


Figure 2-2 (Part 1). File Structure of a Start-Up System

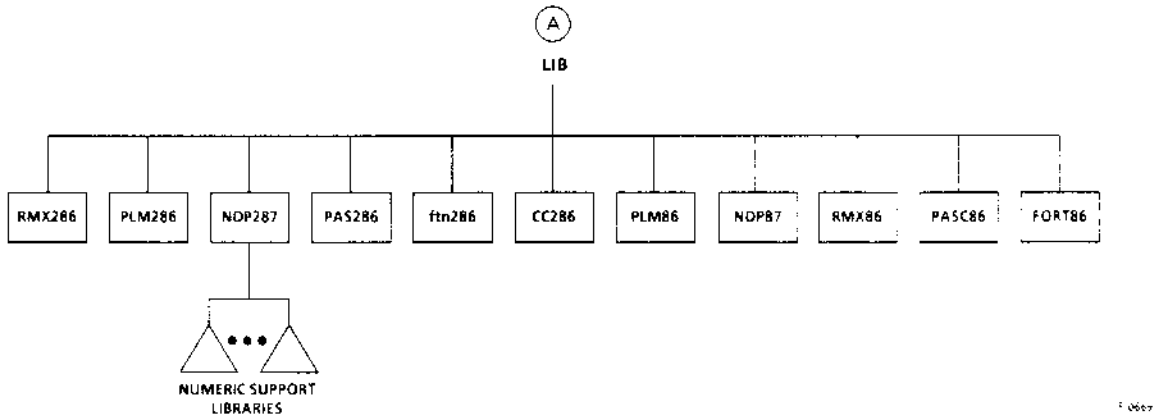


Figure 2-2 (Part 2). File Structure of a Start-Up System

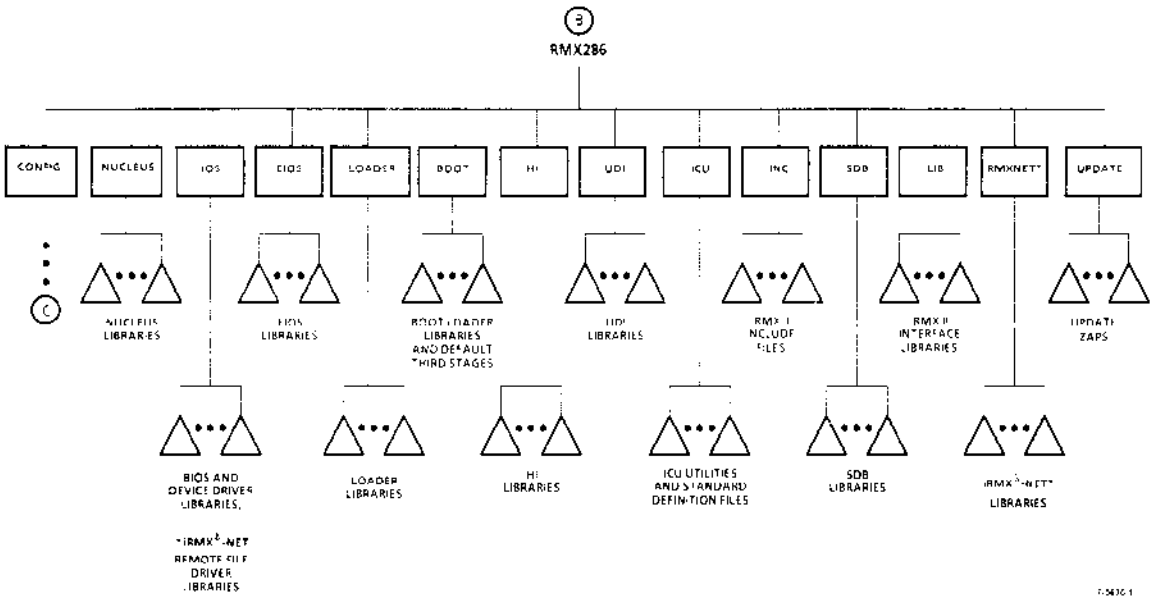


Figure 2-2 (Part 3). File Structure of a Start-Up System

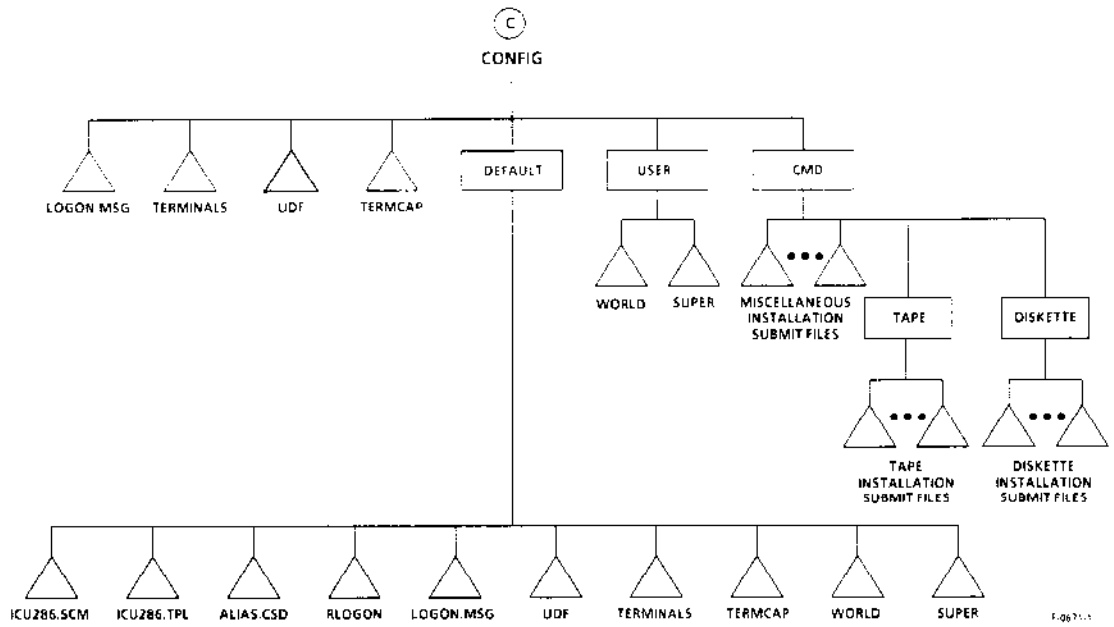


Figure 2-2 (Part 4). File Structure of a Start-Up System

NOTE

Figure 2-3 shows the minimum necessary file structure to allow you to boot iRMX II.

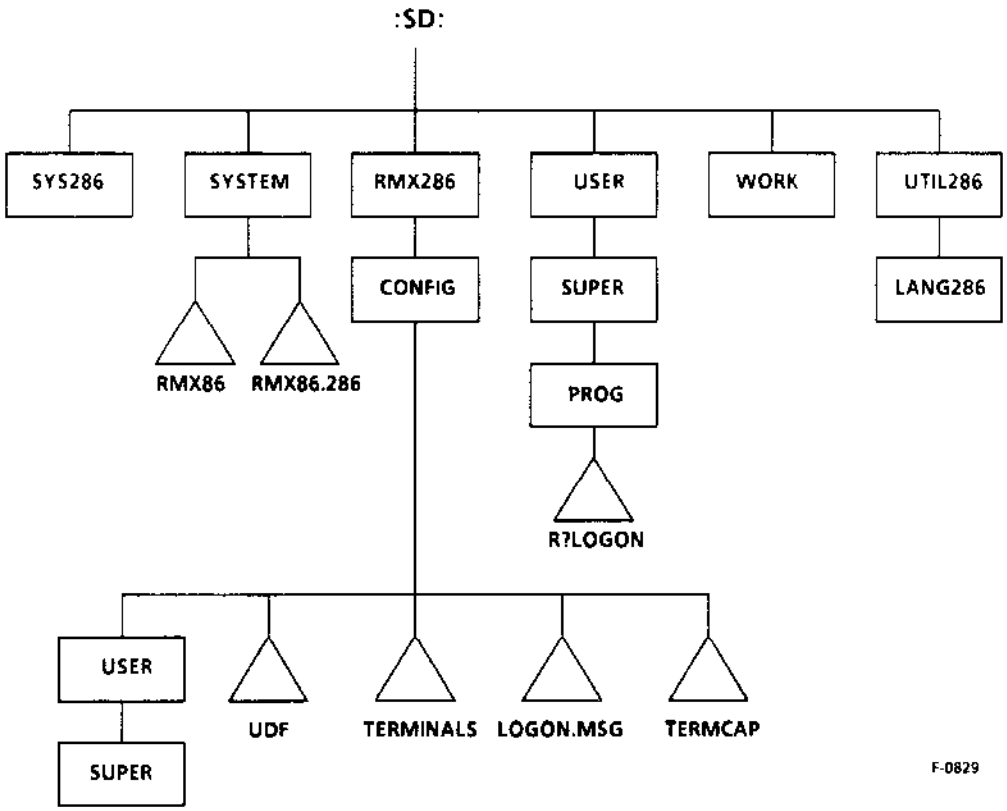


Figure 2-3 Minimum File Structure Needed to Boot iRMX®

2.6.3 Pathnames

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

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

DEPT1/TOM/TEST-DATA/BATCH-2

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

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

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

For example, suppose your starting point in the file tree is the directory DEPT1/TOM shown in Figure 2-1. To access a file in directory BILL from this starting point, you can use the circumflex in the pathname. To indicate file SIM-SOURCE in directory BILL, you could enter the pathname

^BILL/SIM-SOURCE

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

You can use more than one circumflex to go up any number of levels within the file structure. For example, if your starting point is TOM, then you can go up to the root directory by using two circumflexes.

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

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

The initial slash causes the operating system to search in the root directory for directory DEPT1 instead of starting the search in the current starting directory (TOM).

2.6.4 Logical Names

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

- Each logical name must contain between 1 and 10 ASCII characters, excluding the colons surrounding the name.
- The hexadecimal representation of each character must be between 021H and 07FH inclusive (ASCII printable characters).
- The logical name cannot include the characters colon (:), slash (/), up-arrow or circumflex (^), asterisk (*), or question mark (?).
- When you specify a logical name, you must surround it with colons.

When referring to logical names, this manual always lists the surrounding colons. You can use the LOGICALNAMES command to view all the current logical names. The LOGICALNAMES command is described in Chapter 4.

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

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

or

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

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

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

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

2.6.4.1 Creating Logical Names for Devices

Logical names for devices are created in two ways: by invoking the ATTACHDEVICE command (refer to Chapter 4 for details) or configuring the operating system to create them. Section 2.6.4.4 lists the logical names created by the operating system.

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

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

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

:F1:DEPT2/HARRY

You can see that for devices containing named files, the device logical name is actually a logical name for the root directory on that device. Entering the DIR command (described in Chapter 4) as follows lists the root directory of the :F1: device:

DIR :F1:

2.6.4.2 Creating Logical Names for Files

Logical names for files are created in two ways: by invoking the ATTACHFILE command (refer to Chapter 4 for details), or by configuring the operating system to create them. The operating system establishes a number of logical names for files during system initialization. (Section 2.6.4.4 lists these.)

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

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

where :F1: is the logical name for the device that contains the file. You can establish a short logical name for this long pathname, such as :BATCH:. (You could also say that you attached the file with the logical name :BATCH:.) Then, whenever you want to refer to the file in a command, you can specify the logical name instead of the pathname.

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

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

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

:TEST:BATCH-2

2.6.4.3 Where Logical Names are Stored

When the operating system creates logical names at initialization time, or as a result of ATTACHFILE or ATTACHDEVICE commands, it places the logical name, along with a token for a connection to the file or device, into an object directory. This process is referred to as cataloging the logical name (refer to the *iRMX II Extended I/O System User's Guide* for more information about this process). The object directory that receives this information determines the scope of the logical name (that is, who can use the logical name). Object directories fall into three categories:

Root object directory	Logical Names cataloged in the object directory of the root job can be accessed by every user. When you use ATTACHDEVICE to create logical names for devices, the operating system catalogs the logical names in the root directory. Logical names cataloged in the root object directory remain valid until deleted or until the system is reinitialized.
Global object directory	Logical names can be cataloged in the object directory of a job designated as a global job (refer to the *iRMX II Extended I/O System User's Guide* for more information about global jobs). Each interactive job (user session) is a global job. When you use ATTACHFILE to create logical names for files, the operating system catalogs the logical names in your global job. Likewise, if you invoke any commands that issue ATTACHFILE commands (such as a SUBMIT command), the operating system catalogs the logical names in your global job. You (and any commands you invoke) can use the logical names cataloged in your interactive job. However, other users have no access to these logical names. Logical names cataloged in your interactive job remain valid for the life of your interactive job or until deleted. Logical names are also valid in a background environment. When the BACKGROUND command is invoked by the CLI, it creates a global job for the commands invoked within the background environment. All the logical names that were valid when the BACKGROUND command was entered are also valid in the background environment. Logical names can be assigned in either a background or a foreground environment as they are independent of each other.

Local object directory	Logical names can be cataloged in the object directory of the job itself. When you invoke a command (such as DIR), the operating system creates a job for that command and catalogs certain objects in its object directory. A command that you create and invoke might also use iRMX II system calls to catalog logical names in its own object directory. Logical names cataloged in a local job can only be used in the context of that job. They remain valid only until the job exits or is deleted.

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

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

2.6.4.4 Logical Names Created by the Operating System

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

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

:BB:	A device treated as an infinite sink (byte bucket). Anything written to :BB: disappears, and anything read from :BB: returns an end-of-file.
:CONFIG:	A directory in which the Human Interface expects to find user configuration files.
:LANG:	A directory used to store language products, such as assemblers, compilers, and linkers.

:LP:	A logical name for the line printer.
:SD:	The system device. If you used the Bootstrap Loader to load your system, and your system is configured to include the automatic boot-device recognition feature, this logical name refers to the device from which the Bootstrap Loader read the operating system file. You should never change the default logical name for the system device.
:STREAM:	The stream file connection. To create a connection to a stream file, you must use this logical name as the prefix portion of the pathname.
:SYSTEM:	The directory containing the Human Interface commands.
:UTILS:	A directory used to store utility programs such as the iRMX-NET commands.
:WORK:	A directory that Intel language translators and utilities use to store their temporary and work files.

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

:$:	Your default prefix. This is the path to your default directory. If you do not specify a logical name (a prefix) or a '/' at the beginning of a pathname, the operating system automatically uses :$: as the prefix. In this case, the operating system assumes that the file resides in the directory corresponding to :$:. During an interactive session, you can use the ATTACHFILE command to change the directory corresponding to :$:. For instance, the DIR command without any parameters is equivalent to entering "DIR :$:".
:HOME:	Your default prefix when you first start using the Human Interface. Initially, :HOME: and :$: represent the same directory. Using this logical name, you can re-establish your original :$: logical name. Issuing the ATTACHFILE command with no parameters sets :$: equal to :HOME:. You cannot use ATTACHFILE to change the directory corresponding to :HOME:, which is established during system configuration.
:PROG:	A directory in which to store your programs.

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

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

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

On initialization, your Human Interface may create additional logical names. These logical names are configuration parameters. Contact the system manager for more information about the logical names initially available to you. The *Extended iRMX II Interactive Configuration Utility Reference Manual* discusses this subject in more detail.

2.6.4.5 Removing Volumes from Devices

Removing volumes from devices (such as removing flexible disks from drives) destroys any connections that may have existed to files on that device. Therefore, any logical names that represent files on the volume are no longer valid once you remove the volume. The names remain cataloged in the directories, but they do not represent valid connections. Therefore, before removing volumes, invoke DETACHFILE commands (as described in Chapter 4) to detach the files. This removes the invalid names from the directories.

2.6.5 Wild Cards

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

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

FILE1
FILE2
FILEA

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

FILE1
FILE.OBJ
FILE
FILECHANGE

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

IF?.

matches every file containing the sequence "IF" followed by any character and a period. These files could include all of the following files:

RMXIFC.LIB
IFL.P28
LNKIFC.

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

COPY A* TO B*

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

ALPHA
A112
A

The previous command would copy files in the following manner:

ALPHA TO BLPHA
A112 TO B112
A TO B

Note these characteristics when using wild cards:

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

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

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

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

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

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

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

  COPY *1 TO X

  It copies files in the following manner:

  A1 TO X
  B1 AFTER X
  C1 AFTER X

  But if X is a directory, the Human Interface does not concatenate these files. Instead, the Human Interface copies each file over to the new directory. Refer to the "Command Name" section later in this chapter for more information about the prepositions TO, OVER, and AFTER.

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

  COPY *X* TO :PROG:*1*

  This command copies

  ABXCDEFXGH TO :PROG:ABXCDEF1GH

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

2.7 DEVICES

You can communicate with various types of devices (disks, terminals, etc.). Each device supports one or more types of files (named or physical). The list below shows which kinds of devices you can communicate with and the types of files supported on those devices.

Terminals	You need the terminal to communicate with the Human Interface. You can also write programs that read from and write to terminals. Terminals are accessed as physical files.
Disks	Disks provide permanent storage for programs and data. You can communicate with a variety of disk devices: Winchester disks, other hard disks, and flexible disks. During format, backup, restore, and disk verification operations, disks are treated as physical files. Other operations can access named files.
RAM Disk	A RAM disk provides you with an area of memory that acts like a secondary storage device in the system. The RAM disk creates a disk image within your memory and provides faster accessibility to your files. It is similar to a physical disk in all aspects except that it does not provide permanent storage. If there is a power failure or if the system fails for some reason, the data on the RAM disk is lost.
Tapes	Tapes provide a quick and convenient method of backing up hard disks. You can use those tape drives only for backing up and restoring files (via the BACKUP and RESTORE commands). The Operating System does not support tape drives for any other uses.
Bubble Memory	You can reconfigure the operating system to support bubble memory. Once you have done this, you can treat the bubble memory as a physical device, or you can use it as a disk to store named files.

2.8 AUTOMATIC DEVICE CHARACTERISTICS RECOGNITION

Automatic device recognition gives the operating system the ability to recognize and access named disks of different formats without requiring you to reattach the device. This feature does not work with physical files. Automatic device recognition is described briefly in the following sections. Refer to the *Extended iRMX II Interactive Configuration Utility Reference Manual* for more detailed information than is given here.

2.8.1 How Automatic Device Characteristics Recognition Works

The Basic I/O System, the Extended I/O System, and the FORMAT command combine to provide automatic device characteristics recognition as follows:

1. When the FORMAT command formats a flexible diskette as a named volume, it formats it with the same characteristics that match the name you specified when you used ATTACHDEVICE to attach the device. (Characteristics were defined for the name during system configuration.) However, when the diskette is configured to be in "standard" format, it formats track 0 with a fixed density (single density) and a fixed sector size (128 bytes) regardless of the way it formats the rest of the disk. On track 0, FORMAT places the iRMX II volume label, a table that describes the characteristics of the remainder of the volume (granularity, density, number of sides, etc.). Refer to the *Extended iRMX II Disk Verification Utility Reference Manual* for a description of the iRMX II volume label.

   Because track 0 is formatted the same way for all named disks, the Basic I/O System can access the information on track 0 without knowing the format of the remainder of the disk.

2. When you attach a device to the system as a named device (using the ATTACHDEVICE command), you specify the name of a DUIB (device-unit information block) as one of the ATTACHDEVICE parameters. The DUIB names you can use are the ones that are specified as input to the "Device-Unit Information" screen of the ICU.

   The DUIB tells the Basic I/O System which device-unit (disk drive) to attach and which characteristics (granularity, density, number of sides, etc.) to assume about the disk drive.

3. During the attach process, the Basic I/O System reads the information from track 0 (the volume label) and compares it with the information in the DUIB you specified when attaching the device. If the information does not match, the Basic I/O System does the following:

   a. It compares the information in the volume label with all the other DUIBs defined for that device-unit. If it finds a match, it "switches" DUIBs and uses the matching one as the current DUIB.

   b. If none of the DUIBs defined for that device-unit matches the information in the volume label, the Basic I/O System creates a temporary DUIB that does match. It uses the information from the DUIB that you specified when attaching the device and modifies it with information from the volume label. The Basic I/O System names the temporary DUIB by placing a question mark (?) at the beginning of the DUIB name specified when the device was attached.

4. Whenever you remove a disk from a drive (assuming that the drive can recognize when a disk has been removed, such as with 8-inch diskettes), the operating system automatically detaches the device. If it was accessing the device through a temporary DUIB (as opposed to the one you specified as an ATTACHDEVICE parameter), it deletes the DUIB. However, it remembers the name of the DUIB that you specified as input to ATTACHDEVICE.

   If the drive cannot recognize and signal disk changes, such as with 5.25-inch diskettes, the operating system is unable to handle these disk changes automatically.

5. When you insert a new disk into the drive and attempt to access it as a named volume (by invoking the DIR command, for example), the operating system automatically reattaches the device using the same process listed in step 3.

From these steps, you can see that you can continue to change diskettes without having to detach and reattach the device. The operating system makes this change for you automatically. However, this process occurs only for named-file operations. Whenever the operating system performs physical-file operations, it cannot use the temporary or "switched" DUIBs. Instead, it must use the DUIB you specified as a parameter to ATTACHDEVICE.

CAUTION

If the device cannot signal the operating system when a volume changes (such as with 5.25-inch diskettes), the process described above cannot take place automatically. In this case, you must detach the device, insert the new diskette, and reattach the device. Otherwise, chances are you will destroy the file system on the new volume.

2.8.2 Commands That Cannot Recognize Device Characteristics

Because the device characteristics recognition feature does not apply to physical-file operations, the following Human Interface commands cannot use this feature:

FORMAT
BACKUP
RESTORE

Each of these commands must detach the device and reattach it as a physical device, preventing the Basic I/O System from recognizing the characteristics of the volume. These commands assume that the device characteristics are those listed in the DUIB you specified as an ATTACHDEVICE parameter. If you do not include, for example, a DUIB for a double-sided, double-density disk in your configuration, you cannot format such a disk. You cannot create a backup volume in this format or restore information from one.

If you plan to use one of these commands and you are not sure how your device was attached, use DETACHDEVICE and ATTACHDEVICE to reattach the device with the characteristics you require.

2.8.3 Operational Considerations for MSC Devices

If your system contains an 8-inch floppy disk drive and an Mass Storage Controller (MSC) controller, you may receive error messages that are not appropriate when switching diskettes. For example, if you attach your device as a double-sided, double-density device and insert a single-sided, single-density diskette, you will receive an I/O error message when attempting an I/O operation. In this situation, the message does not indicate a problem. If you try the I/O operation again, it will usually succeed.

2.9 HUMAN INTERFACE COMMAND SYNTAX

This section describes the general syntax rules that apply when entering Human Interface commands (not CLI commands) at a terminal. These rules apply equally to both the supplied Human Interface commands and any user-created commands that may have been added to your system. The command descriptions in Chapter 4 contain more information about these commands.

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

command-name [inpath-list [preposition outpath-list]] [parameters] <CR>

where

command-name	Pathname of the file containing the command's executable object code.
inpath-list	One or more pathnames, separated by commas, of files to be read as input during command execution.
preposition	A word that tells the executing command how to handle the output. The four prepositions used in Intel-supplied commands are TO, OVER, AFTER, and AS.
outpath-list	One or more pathnames, separated by commas, of files to receive the output during command execution.
parameters	Parameters that cause the command to perform additional or extended services during execution.
<CR>	A line terminator character. This character terminates the current line and causes the cursor to go to a new line. This character also causes a command to be loaded and executed if the <CR> character is not preceded by the ampersand (&). The RETURN (or CARRIAGE RETURN) key and NEW LINE (or LINE FEED) key are both line terminators.

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

continuation mark	An ampersand (&) indicates that the command continues on the next line. When you include the ampersand, the Human Interface displays two asterisks (**) on the next line to prompt for the continuation line. All characters appearing after the continuation mark but before the line terminator are interpreted as comments. Within available memory limits, you can use as many continuation lines for a given command as you desire. After you enter the line terminator without a preceding ampersand, the invoked command receives the entire command string as a single command.
comment character	A semicolon (;) indicates that all text following it on the current line is a nonexecutable comment. You can also enter comments after a continuation mark (&) but before the line terminator. A common use of comments in commands is in a SUBMIT file (see the SUBMIT command in Chapter 3).

quoting characters	Two single-quotes (') or double-quotes (") remove the semantics of special characters they surround. For example, if you surround an ampersand (&) with single quotes, the ampersand is not recognized as a continuation character. The same holds for other characters such as asterisk (*), question mark (?), equals (=), and semicolon (;). The only special characters not affected by the quoting characters are the pathname separators (/) and dollar sign ($).

Although you can use either single quotes or double quotes as quoting characters, you must use the same quoting character at the beginning and at the end of your quoted string. If you want to include the quoting character inside your quoted string, you must specify the character twice. For example

```
'can''t'
```

You can accomplish the same effect by using the other quoting characters follows:

```
"can't"
```

Although the Human Interface places no restriction on the number of characters in a command, each terminal line can have a maximum of 76 characters, including any punctuation, embedded blanks, continuation marks, nonexecutable comments, and line terminator. If your command requires more characters, use continuation lines.

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

2.9.1 Command Name

Each Human Interface command is a file of executable code that resides in secondary storage. When you specify a command name, you actually specify the name of the file containing the command's code. If you write your own command (refer to the *Extended iRMX II Human Interface User's Guide* for information), you invoke it by entering the name of the file that contains it. After you invoke a command, the operating system loads it from secondary storage into memory and executes it with the parameters you specify.

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

- If you enter the complete pathname of the command (that is, if you include a logical name as the prefix portion of the pathname), the operating system searches only the device and directory you specify for the command. If it cannot find the command there, it returns an error message.

- If you enter only the last component of the pathname (such as COPY instead of :SYSTEM:COPY), the operating system automatically searches previously designated directories for the command. It does not return an error message until it has searched each of the directories. The number of directories searched and the order of search are determined during the Human Interface configuration. The standard search path of directories is:

  :PROG:
  :UTILS:
  :SYSTEM:
  :LANG:
  :$:

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

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

```
:SYSTEM:COPY
```

Using the ALIAS command (see Chapter 3) provides a faster more direct means of retrieving a command from one of the directories. To improve performance and create a shorter name, you might define the Human Interface ATTACHFILE command with an alias

```
alias af =:SYSTEM:ATTACHFILE
```

In this case, every time you enter AF, the operating system replaces it with :SYSTEM:ATTACHFILE and invokes the ATTACHFILE command found in the :SYSTEM: directory.

2.9.2 Prepositions

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

TO

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

```
<pathname>, already exists, OVERWRITE?
```

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

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

OVER

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

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

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

AS	A special preposition used with the ATTACHDEVICE and ATTACHFILE commands. When you use the AS preposition, the operating system does not assume that the command contains input pathnames and output pathnames. Rather, it sees the parameters as entities that it must associate. For example, ATTACHFILE associates a pathname with a logical name as follows:

2.9.3 Inpath-List and Outpath-List

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

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

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

Table 2-1. Input Pathname and Output Pathname Combinations

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

2.9.3.1 One-for-One Match

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

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

2.9.3.2 Concatenate

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

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

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

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

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

2.9.3.3 Error Conditions

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

2.9.4 Other Parameters

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

For example, the DIR command (described in Chapter 4) lists the contents of a directory. You can enter several different keyword parameters to specify the amount of information displayed and the format of the display. A command such as

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

The command descriptions in Chapter 4 list the keyword parameters available with each command. Although the descriptions list the complete names for the keywords, you need enter only enough characters to uniquely identify the keyword. For example, you could enter the previous command as

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

Some keyword parameters also require an associated value. An example of this is the FORMAT command (described in Chapter 4), which prepares secondary storage volumes for iRMX II use. A command such as

formats a volume on device :F1: and sets the number of files to 10. The keyword in this command (FILES) has an associated value (10). Although this example and the descriptions in Chapter 4 use the equal sign (=) to associate keywords and values, you can do this two ways:

```
keyword = value
keyword (value)
```

The blanks are optional; you can use either method to enter Human Interface commands.

2.10 SYSTEM MANAGER

The multi-user Human Interface supports a user called the system manager. The system manager maintains the multi-user configuration files. The system manager can modify these files to add or delete user IDs, add or delete terminals, and change terminal or user characteristics (refer to the *Extended iRMX II Interactive Configuration Utility Reference Manual* for more information). For security reasons, no user other than the system manager can write to these files. (Although user passwords are kept in one of these files, :CONFIG:UDF, and other users can read that file, the passwords are encrypted and there is no way to decrypt them.)

The system manager has a special user ID that gives privileges that other users do not have, such as

- Change the access rights of any file, regardless of the file's owner.
- Read access to all data files and list access to all directories, unless specifically denying himself or herself those access rights.
- Detach devices attached by any user.
- Delete any user from the system.

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

2.11 ADDING USERS TO THE SYSTEM

To function correctly, a Human Interface system requires information about all users (operators) and terminals that intend to access the system via the Human Interface. You can supply information about one of these users via the ICU during the configuration of the Human Interface. This user is called the resident user, and the process of configuring it is called resident user configuration (refer to the *Extended iRMX II Interactive Configuration Utility Reference* for information on resident user configuration). Other users, if any, and their terminals must be defined in iRMX II files that you set up during system installation. These users are called non-resident users, and the process of configuring them and their terminals is called non-resident user configuration. The files are called non-resident configuration files.

The system manager (who has user ID 0) can modify these files to add users or terminals, delete users or terminals, or change characteristics of users or terminals. Depending on the type of modifications made, the changes take effect either the next time the affected user logs onto the system or the next time the system is initialized. To prevent unauthorized users from changing the system configuration, the system manager should be the only user with change access to these files.

This section describes the types of system terminals you can configure, defines the names you must use for the non-resident configuration files, and describes the information you must place in these files. It also provides an example of adding a non-resident user to your system.

2.11.1 System Terminals

With the configuration files described in this chapter, you can assign Human Interface terminals to be one of two kinds: static logon terminals or dynamic logon terminals. The following sections discuss these terms further.

2.11.1.1 Static Logon Terminals

Each static logon terminal is configured to service a specific operator. Therefore, the logon process is automatic and invisible to the operator. When the Human Interface starts running, it has information about the operator, such as the user ID, the amount of memory available to this operator, and the interactive job's maximum priority. To change the Human Interface specifications for a static logon terminal, you must change the non-resident configuration files and reboot the operating system.

2.11.1.2 Dynamic Logon Terminals

Dynamic logon terminals can service many different operators on a request-by-request basis. To determine which operator wants access to the operating system via a dynamic logon terminal, the Human Interface requests information before allowing the operator to access the system. This information consists of a logon name and a password. The Human Interface verifies that the information entered is valid by checking the non-resident configuration files. Then it sets up the terminal based on other information listed in those files.

To change the Human Interface specifications for a user who accesses the system from a dynamic logon terminal, you must change the non-resident configuration files. The changes take effect the next time the user logs on.

Each user that logs onto the system is allocated a memory partition. If the Human Interface cannot allocate the amount of memory requested, but it has enough memory to activate the user's initial program, the Human Interface allocates all the free memory available and displays this warning on the user's screen:

```
*** WARNING:    The system cannot provide the minimum memory you
***             requested.  You will come up with all the memory
***             that is currently available in the system.  If this
***             is a problem, contact the System Manager.
```

If there is not enough memory for the user's stack, the following message is displayed:

```
*** HI LOGON ERROR:   Insufficient user memory available at this time.
***   Try to logon again later.
```

2.11.2 Non-Resident Configuration Files

Before non-resident users can access the Human Interface, you must prepare four configuration files that describe all users and terminals. The files are

:CONFIG:TERMINALS	The terminal-configuration file. This file contains the device names of the terminals, the names of the static users (if any) that are associated with each terminal, and the terminal type. There is only one terminal-configuration file per application system, and its name is fixed. It contains information about all the terminals in the system.
:CONFIG:TERMCAP	The terminal definition file. This file defines each terminal type with its configuration commands and default values. Included in this file is a string of ASCII characters defining the terminal name. To use a specific terminal name in the :CONFIG:TERMINALS file, it must have been previously defined in this file. There is only one terminal definition file per application system. For detailed information on defining the :CONFIG:TERMCAP file, see Appendix B
:CONFIG:UDF	The user-definition file. This file contains the names of the users that can log onto static or dynamic terminals, their user IDs, and their passwords (if any). There is only one user-definition file per application system, and its name is fixed. It contains information about all non-resident users in the system. This file has a non-standard formatting that allows users to access, over the iRMX-NET network, microcomputer systems running the Xenix operating system. Therefore, the only way to add or delete information in this file is to use the Human Interface PASSWORD command.
:CONFIG:USER/username	The user-attribute files. Each of these files lists iRMX II-specific information about a user. This information includes the user's minimum and maximum partition size, maximum task priority, default prefix, and initial program. There is one user-attribute file for each non-resident user in the system. These user-attribute files must be in the :CONFIG:USER directory and the name of the file must be identical to the user (logon) name of the corresponding user.

Each of these configuration files resides on the system device in the :CONFIG: directory. You specify the pathname of this directory during the configuration of the Extended I/O System. Changing the definition of the :CONFIG: logical name later has no effect on the configuration files checked, because the Extended I/O System (and the Human Interface) use the complete pathname as specified during EIOS configuration.

The following paragraphs describe :CONFIG:TERMINALS, :CONFIG:UDF, and :CONFIG:USER in detail.

2.11.2.1 Terminal Configuration File (:CONFIG:TERMINALS)

The terminal-configuration file (:CONFIG:TERMINALS) defines all terminals through which non-resident users intend to access the system through the Human Interface. It contains several lines of information, as follows:

1. An integer indicating the number of terminals to be connected. This is the first line of the file.
2. Device name and attributes of the terminals, one line of information for each terminal.

The device name and attributes of a terminal must reside on a single line, with commas separating the individual elements. Embedded blanks within a name are not allowed. The format of this information is as follows. (Square brackets ([ ]) indicate optional elements. Spaces are allowed between elements.)

```
device-name,[user-name],reserved,[terminal-name]
```

where

device-name	Name of the terminal. This name must be the physical device-unit name (DUIB) specified for the terminal during configuration.

user-name

The presence or absence of this parameter indicates whether the terminal is a static logon terminal or a dynamic logon terminal, as follows:

- If this parameter is present, the terminal is a static logon terminal. The parameter must be a string of three to eight ASCII characters specifying a user name. When the Human Interface starts running, it automatically logs this user onto the system from this terminal.

  For any name you specify here, there must be a corresponding entry in the UDF and a corresponding user-attribute file available.

- If this parameter is absent, the terminal is a dynamic logon terminal. Before users can access the Human Interface from this terminal, they must supply a user name and password during the logon process.

reserved

Reserved for future use. Enter a null value (comma or line terminator used as a place holder).

terminal-name

An ASCII string equivalent to the terminal name in the :CONFIG:TERMCAP terminal definition file. ANY is the default value. The ANY option applies to all terminal types, however, it has a limited number of functions. Its main purpose is to provide compatibility with iRMX II.1. To get all the added editing features of iRMX II.2 and II.3, you can change this value to a specific terminal name, such as VT100. :CONFIG:TERMCAP defines the following terminal names:

- 1510E
- 1510T
- ADM3A
- QVT102
- TV910P
- TV950
- VT100
- VT102
- VT52
- WYSE50
- ZENTEC

If you omit the user-name parameter but specify the terminal-name parameter, you must include two commas as a placeholder for the omitted parameter.

When the Human Interface starts running, it assigns memory for all static logon terminals in the order they are defined in the terminal-configuration file. Therefore, you should define the terminals in order of their importance, to guarantee that the high priority terminals have access to the system.

The following is an example of a terminal-configuration file that defines two static logon terminals and one dynamic logon terminal:

```
3
T1,ted,,VT100
T3,,ZENTEC
T4,kim
T6
```

Terminals T1 and T4 are static logon terminals associated with users ted and kim. The sizes of their memory partitions are determined from their user-attribute files. Ted is assigned to terminal T1 which is a VT100 terminal. Terminal T3 is a dynamic logon terminal.

Terminal T6 is not included in the list of terminals the Human Interface initializes because it is the fourth terminal in the list. The first line of the terminal-configuration file indicates that only three terminals are present.

2.11.2.2 User-Definition File (:CONFIG:UDF)

This file defines the names, user IDs, and passwords of all non-resident users (those who access the Human Interface from static or dynamic logon terminals). The passwords maintained in this file are encrypted to prevent unauthorized users from obtaining privileged information by viewing this file.

To accommodate the iRMX-NET network, in which both iRMX II systems and microcomputer systems such as the iRMX I and Xenix operating systems can share files on the same network, the format of the user-definition file (UDF) has the same format as the Xenix password file. Passwords are encrypted in the same manner, and each line of the file ends in a line feed only (not a carriage return/line feed pair). This common format enables operators to log onto either the iRMX I, iRMX II, or Xenix operating systems using the same user name and password.

Because the UDF requires a special format (encrypted passwords and nonstandard line terminators), ordinary text editors cannot be used to create this file. Instead, the system manager must use the Human Interface PASSWORD command to add entries to the file. PASSWORD automatically encrypts passwords, and sets up the appropriate line terminators.

In brief, the system manager can specify the following information about each user with the PASSWORD command:

user name	A three- to eight-character name that the user supplies when logging onto the system.
password	A zero- to eight-character, case-sensitive password that the user must also supply when logging onto the system. If this user is associated only with static logon terminals, the password is optional. In other cases, a password is required. A null password indicates that the user does not need to enter a password during logon, just a carriage return when the password prompt appears. If the system manager specifies the characters "NO LOGIN" as the password, the associated user cannot log onto the system from a dynamic logon terminal. (This "NO LOGIN" password can be assigned to static logon users to prevent them from logging onto dynamic logon terminals.) The PASSWORD command automatically encrypts the password before placing it into the UDF.
user ID	Decimal number in the range 0 through 65535. This number represents the user's ID. The ID 65535 represents the WORLD user. ID 0 represents the system manager. IDs 32768 through 65534 are reserved for future use by the operating system.
group ID	A second type of user ID in the range of 0 through 65535. This parameter is optional. If you enter both a user ID and a group ID, the system assigns a user ID, a group ID and the WORLD ID. If you do not enter a group ID, the system assigns the user ID and the WORLD ID.

Other fields in the UDF are reserved for Xenix systems or for future use.

2.11.2.3 User-Attribute Files (:CONFIG:USER/username)

The UDF doesn't contain all the information the Human Interface needs to set up a user. The remaining information is contained in files in the :CONFIG:USER directory. Each user must have a separate file (the name of the file is the same as the user's logon name). Each file contains user information (called attributes). The attributes must be separated by commas; embedded blanks are not allowed. The format of each user definition file is as follows (attributes may be entered on one line):

minimum partition-size,
maximum partition-size,
max-task-priority,
default-prefix-pathname,
initial-program

where:

minimum partition-size	Decimal number specifying the minimum size, in 1024-byte (1K) units, of the memory partition that the Human Interface assigns to the interactive jobs for this user. The Human Interface assigns a memory partition of this size when the user logs onto a terminal that uses dynamic memory partitions.
maximum partition-size	Decimal number specifying the maximum amount of memory, in 1024-byte (1K) units, the user job can have. The difference between the maximum and minimum values is borrowed from the parent, if necessary.
max-task-priority	Decimal number in the range 0 through 255. This number specifies the maximum priority (lowest numerically) that any task associated with this user can have. Intel highly recommends that you assign a priority value greater than 141 (assign yourself a number less than 141).
default-prefix-pathname	Pathname of the directory that serves as this user's default prefix (corresponding to the :HOME: and initial :$: directory). This directory is normally the :SD:USER/username directory, where username is the same as the name of the user. The directory specified in this field must exist or the user will be unable to access the Human Interface.

initial-program	Pathname of the file containing the user's initial program. This is the program that begins running immediately after the user logs on. If you omit this value or enter :SD:SYS286/CLI, the Human Interface uses its standard command line interpreter (CLI) as the initial program. If you want to continue using the Release 1.0 CLI , enter :SD:SYS286/R1CLI in this field.

The following is an example of a user-attribute file:

pathname	contents
:CONFIG:USER/STEVE	200,800,150,:SD:USER/STEVE,:SD:SYS286/CLI

2.11.3 Setting up a Protected Environment

With a single-user Human Interface, there is no need to restrict access to files or devices. There is only one user, and that user requires access to all files and all devices in the system. However, with a multi-user Human Interface, file and device access becomes a much more important issue. For example

- A multi-user Human Interface requires a terminal-configuration file, a terminal definition file, a UDF, and user-attribute files. To maintain system security and integrity, you should limit access to these files.
- Many users might want to prohibit other users from viewing the files in their default directories. However, some users might want to grant other users the ability to access those files. To allow for this, users should be the owners of their default directories.
- Many users should be able to run Human Interface commands and utilities (such as compilers or editors). To do this, they require the ability to list the directories containing the commands and utilities and the ability to execute and read the commands and utilities themselves. However, to protect the files from damage, you should limit the number of users who are able to modify or delete those files.
- Some devices (such as a hard disk or the device that serves as the system device) should be available to all users. However, to protect users who access these devices, only one user (the system manager) should be able to detach the devices.

To create a multi-user system that provides all these protection features, the system manager must set up the correct file structure before allowing other users to access the system. The installation of the iRMX II Operating System, as described in the *Extended iRMX II Hardware and Software Installation Guide*, sets up the protected environment described here, enabling the system manager to maintain control of the application system.

2.11.4 File Structure of a Protected Environment

There are many ways to set up a file structure to support a multi-user Human Interface. The ICU and the definition files provide options to support many different configurations. However, unless you have specific reasons for setting up a special structure, you should use the standard file structure described in this section. This standard file structure is created when you install the Operating System. All of the Updates to the operating system also assume this standard file structure.

Figure 2-2 shows the standard directories and data files that exist on a system device after you have completed the installation of an iRMX II Start-up system. The figure also lists the user ID of each file's owner and the access rights of WORLD. The owner is important because that user always has the ability to change the access rights associated with the file or the directory. WORLD access is important because it indicates that all users have access rights. The Human Interface always ensures that the WORLD ID is included in the ID's that define a Human Interface user.

In the figure, :SD: indicates the root directory of the system device. You establish this logical name for the default system device via input to the ICU during configuration of the Extended I/O System. If you specify automatic boot device recognition, and if you use the :SD: logical name when specifying other files and directories during configuration (such as when specifying the default prefixes in the user attribute files), you can bootstrap load the system from any device. The "Automatic Boot Device Recognition" feature is described in the *Extended iRMX II Interactive Configuration Utility Reference Manual* and in the *Extended iRMX II Bootstrap Loader Reference Manual.*

NOTE

For systems that include the Human Interface, you must assign the logical name :SD: as the name of the system device during EIOS configuration. Assigning a different logical name causes some Human Interface commands to function incorrectly.

During the configuration of the Extended I/O System, you should specify the system manager as the owner of the :SD: device. This prevents users other than the system manager from detaching the system device. Before allowing users to access the system, you should ensure that only user 0 has all access rights to the root directory. Other users should have list access (L) to allow them to view the files in the root directory. However, they should not have change entry (C) or delete (D) access to the root directory. Add entry (A) is optional. The correct access rights are established during installation. (Refer to the *Extended iRMX II Hardware and Software Installation Guide* for more information.)

Three first-level directories listed in Figure 2-2 are used to store commands and utilities. They are LANG286, UTIL286, and SYS286. The system manager should be the owner of these directories and the files they contain. To protect the commands and utilities from damage or deletion, other users should have only list access to the directories (to be able to see what is available) and read access to the data files (to be able to run the commands). If you have the iRMX I Operating System on the same volume, the directories LANG, UTILS, and SYSTEM have the same functions as LANG286, UTIL286, and SYS286, but with iRMX I equivalents.

The directory SYSTEM is present whether you have both iRMX I and iRMX II Operating Systems on the same volume or not. This directory contains the bootloadable system since, by default, the Bootstrap Loader loads from /SYSTEM/RMX86. If you want to change the default, you can configure the Bootstrap Loader to load any other file, or you can specify the file explicitly during invocation of the Bootstrap Loader. In this case, the directory SYSTEM may be deleted.

The first-level directory BOOT is where all Intel-supplied ICU definition files place the generated bootloadable system. You should place files containing new versions of the operating system into this directory. When you use the ICU, you specify the location of these files with the "RAF" prompt on the Generate File Names screen. Specify the name /BOOT/your file name.286 for your boot file.

The first-level directory USER contains the directories that are the default prefixes of the users. The system manager should be the owner of USER, and other users should have only list (L) access. However, each directory contained in USER should be owned by the user with the corresponding user name and should not be accessible by other users. By owning the directory that serves as its default prefix, each user can change the access rights of the directory. This ability enables a user to decide what access rights other users should have to the first user's files. However, because no other users have automatic access to the default directory, the first user can maintain privacy if desired.

Another first-level directory, RMX286, contains all the files necessary to configure your own version of the operating system. It also contains the CONFIG directory which includes the terminal-configuration, user-definition, and user-attribute files. The system manager should be the owner of the directory and the files contained in it. Other users can have list or read access to the subdirectories and files it contains, but they should not be able to add, modify, or delete any information. This prevents everyone but the system manager from modifying the terminal and user configuration.

2.11.5 Adding a Non-Resident User to a Multi-User System

This section provides an example of how to add new users to a multi-user application system. It assumes that the directory structure of the system device is set up as shown in Figure 2-2.

The directories that are important when adding new users are the :SD:USER directory, the :SD:RMX286/CONFIG directory (also accessed by the logical name :CONFIG:), and the :SD:RMX286/CONFIG/USER directory (also accessed as :CONFIG:USER). The :SD:RMX286/CONFIG directory can be changed to any other directory during configuration. Using the "EIOS" screen of the ICU you can define any pathname as the directory for the non-resident user configuration information. For more information see the *Extended iRMX II Interactive Configuration Utility Reference Manual.*

The :SD:USER directory contains the default prefixes (:HOME: directories) of all the users in your application system. As part of adding a new user, you can use the PASSWORD command to automatically create a subdirectory in the :SD:USER directory that has the same name as the logon name of the user you are adding. This subdirectory is the user's new home directory.

The :CONFIG: directory contains the terminal-configuration file (:CONFIG:TERMINALS), the user-definition file (:CONFIG:UDF), and a subdirectory that holds the user-attribute files (:CONFIG:USER). As part of adding a new user, you must modify the user-definition file and add a new user-attribute file to the :CONFIG:USER directory. You can use the PASSWORD command to do this for you. The user-attribute file must have the same name as the logon name of the user you are adding.

As an example, let's add a user to the system, assuming the following information about that user:

Type of user:	Able to access the Human Interface from any dynamic logon terminal.
Logon name	jean
Password	HOME
Group ID	03
User ID	15
Minimum partition size	200K
Maximum partition size	600K
Task priority	200
Default prefix pathname	:SD:USER/JEAN
Initial program	Human Interface CLI

Perform the following steps to add this user to the system, enabling him to log on from any dynamic logon terminal:

1. Log onto the system from a Human Interface terminal.
2. If you are not the system manager (user ID 0), invoke the SUPER command to become the system manager. (This step requires that you know the system manager's password.) Enter the following command:

```
- super
ENTER PASSWORD: <enter password here>
SUPER-
```

3. Invoke the PASSWORD command to create the file :SD:USER/JEAN and add a new entry for this user in the UDF, as follows:

```
SUPER- password
```

The command displays the following information and waits for your response:

```
The following commands are available:
A - Add a user
D - Delete a user
L - List the UDF
C - Change the password
Q - Quit
E - Exit
Enter the command: a
```

The command then displays several prompts. Respond as follows:

```
Enter the user name - jean
Enter the new password - HOME
Repeat the new password - HOME
Enter the user ID - 15
Enter the group ID - 03
Enter the comment - <cr>
Enter the default Xenix directory - <cr>
Enter the default Xenix shell - <cr>
```

The command summarizes the information and prompts you to continue, as follows:

```
User Name = jean
User ID = 15
Group ID = 03
Comment =
Default Xenix directory =
Default Xenix shell =
Do you want to add this user to UDF? y
```

Respond with "y". PASSWORD updates the copy of the UDF it maintains in memory (the permanent copy is updated when you invoke the Exit command), and displays this message:

```
Do you want to create the user directories? y
```

Answering "y" causes PASSWORD to automatically update your user file :SD:USER adding a subdirectory JEAN and a subdirectory under it PROG (:SD:USER/JEAN/PROG). It copies the file :CONFIG:DEFAULT/RLOGON to R?LOGON and creates an empty R?LOGOFF file in the :SD:USER/JEAN/PROG directory. The logon file contains DATE and TIME commands and the command that invokes the ALIAS.CSD file. Thus, each time jean logs onto the system, DATE, TIME, and ALIAS.CSD execute.

After the files are updated, you are prompted for the pathname of the initial program as follows:

```
Initial-program pathname =
```

If you are using the standard CLI, enter a carriage return. After adding the new user, PASSWORD responds with

```
Default Initial Program is RMX286 HI CLI

Added user JEAN
```

The PASSWORD main menu then reappears. Enter the E command to save your changes, as follows:

```
The following commands are available:

A - Add a User
D - Delete a User
L - List the UDF
C - Change the password for the system manager
Q - Quit
E - Exit

Enter the command: e

Updating the master UDF .........Done

SUPER-
```

4. Exit the SUPER command, as follows:

```
SUPER- exit
```

The user jean is now added to the system. Jean can go to any dynamic logon terminal and log onto the application system, using the logon name and password assigned.

Intel®

CHAPTER 3
CLI COMMANDS

3.1 INTRODUCTION

This chapter presents the CLI commands (not to be confused with the Human Interface commands) in alphabetical order. CLI commands are part of the standard Human Interface Command Line Interpreter, and are executed directly by the CLI instead of being loaded from a file and executed as part of the Human Interface. If you have written your own initial program or are using the Release 1 CLI, you may wish to skip this chapter as these commands are only available with the standard CLI.

The commands LOGOFF, SUBMIT, and SUPER can be both CLI and HI commands. If you are using the standard CLI, these commands function as CLI commands and recognize all the CLI features. If you are using your own initial program, these commands are also available as Human Interface commands, but without the CLI features such as aliasing, advanced line-editing, and background processing.

3.2 COMMAND SYNTAX

CLI commands have a general syntax that you should follow when entering them at the terminal. The elements that form a standard command entry include a command name, required input parameters (if any), and optional parameters. The structure of a command line is as follows (brackets [ ] indicate optional elements):

command-name [parameters] [I/O redirection] <CR>

where

command-name	Name of the command.
parameters	Define the subject on which the command is to perform during execution. For example, when you use the SET TERMINAL command, the CLI must know which terminal is being defined.
I/O redirection	Causes the command's :CO: and :CI: to be replaced by the file specified. This parameter can be anywhere after the command name.

<CR>	A carriage return indicating the end of the command.
<escape>	An additional method of invoking the command line entered at the terminal.

3.2.1 I/O Redirection

You can use I/O redirection with both CLI and HI commands. It causes the CLI to replace the commands :CO: and :CI: with the specified file. The operator's terminal (:EO:) remains the same. All error messages, user inputs, and program outputs are transmitted to the new files specified rather than to your terminal. This option is particularly useful when executing the BACKGROUND command. By redirecting your output messages to a file, you free the terminal for other operations.

To use I/O redirection, you should include either or both of these parameters anywhere in the command line. The parameters are formatted as follows and may be in any order:

<infile >outfile

where

infile	The name of the input file that replaces the terminal as standard input.
outfile	The name of the output file that replaces the terminal as standard output.

The angle brackets (<,>) have a specific meaning to the CLI. If you wish to use them for anything other than I/O redirection, include them either in single quotes (') or double quotes ("). The CLI then passes them to the Human Interface as is.

The examples below illustrate the use of the I/O redirection feature.

1. This example uses I/O redirection with the BACKGROUND command to redirect output to a file called COPY.LOG.

2. This example uses I/O redirection to change the source of input from the keyboard to a file named IN.DAT and to redirect the output to a file named OUT.DAT.

3.2.2 Syntax Conventions

You can enter all elements of a command line in uppercase characters, lowercase characters, or a mix of both. The CLI makes no distinction when it reads the commands. The CLI does not recognize continuation marks, comment characters, or quotation marks within its commands. These characters, however, are recognized by Human Interface commands. If the result of a CLI command causes the execution of an HI command, the HI command is governed by HI command syntax rules. For example, this may occur when executing the BACKGROUND command as follows:

Since executing BACKGROUND causes the Human Interface COPY command to be executed, the rules for Human Interface commands apply. This means that a semicolon is recognized. You might compare this to a telephone exchange. First, you dial the CLI, using CLI rules, then, if necessary, the CLI transfers you to the Human Interface command which has its own set of rules.

3.3 COMMAND SYNTAX SCHEMATICS

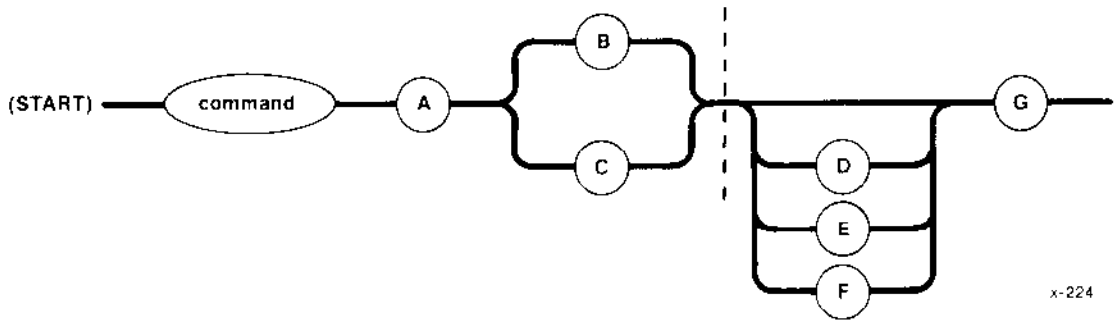
The syntax for each command described in this chapter is presented in a "railroad track" schematic, with syntactic elements scattered along the track. Your entrance to any schematic is always from left to right, beginning with a command name entry.

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 in either uppercase or lowercase; the CLI makes no distinction between cases. Syntactic elements shown in lowercase characters are generic terms, which means that you supply the specific item, such as the pathname for a file.

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

"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 track or the other but not both.

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.



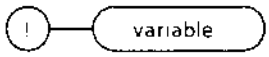
In this example

- A is a required element. It is position-dependent; it must be entered first.
- Either B or C is required. These elements are also position-dependent. Whichever element you enter must follow A immediately.
- D, E, or F are all optional but only one can be selected. These are position-independent elements. If you select one of these elements, you can enter it before or after G.

Table 3-1. CLI Command Dictionary

Command	Function	Page
!	Recalls a specified command line.	3-6
ALIAS	Assigns an abbreviation to a command.	3-8
BACKGROUND	Causes a command to be executed in background mode.	3-13
CHANGEID	Changes the current user ID to any value between 0 and 65535.	3-16
DEALIAS	Cancels the abbreviation assigned by ALIAS.	3-18
EXIT	Leaves the SUPER mode.	3-20
HISTORY	Recalls the last 40 lines entered at the terminal.	3-21
JOBS	Displays a list of background jobs by their job identification number.	3-24
KILL	Cancels a background job.	3-25
LOGOFF	Ends a user session at a dynamic logon terminal.	3-26
SET	Alters the CLI environment by allowing on-line changes to the terminal name, the minimum and maximum background memory pool size, the memory for alias tables, or the prompt string.	3-27
SUBMIT	Reads, loads, and executes a string of commands from a secondary storage file instead of from the keyboard.	3-31
SUPER	Changes the operator to the system manager by changing the user ID.	3-36

This command recalls a specific command line. It has the following format:



F-0508

INPUT PARAMETER

variable	The command line number (in decimal) or prefix of the command to be recalled. The command line number must not be greater than 999.

DESCRIPTION

By entering the ! command, you can recall a specific command line by either its number or its prefix. The CLI searches its history buffer from the most recently entered command, going backward until it finds the specified line number. It then replaces the current command line with the command that was located. You can edit the recalled line, if you wish. If you are recalling a command prefix rather than a line number, be sure to enter enough of the command prefix to make it unique. The CLI recalls the first command line it encounters with the specified command prefix. For example, if you enter

and you have previously entered

```
FORMAT :d:
FTN286 myfile.f28
```

the CLI will display FTN286 myfile.f28 . If you want to recall FORMAT :d:, you should enter

To recall a command line by its number, for example line 29, enter

ERROR MESSAGES

- `<number>, history number not found`

  The number you entered cannot be found in the history buffer.

- `<number>, history number out of range`

  The number you entered is greater than 999.

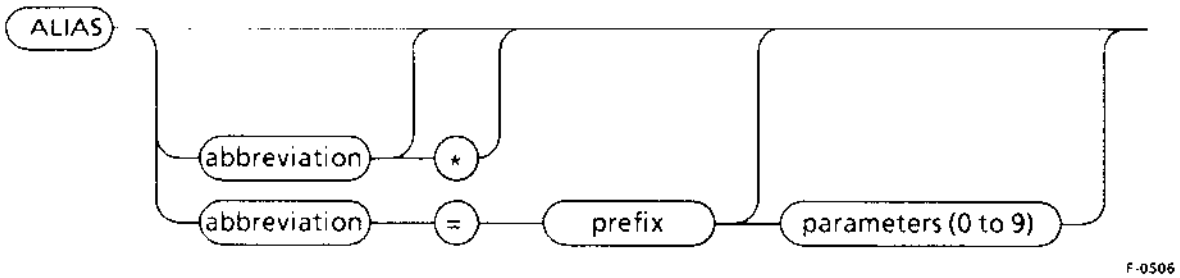
- `<number>, illegal history number`

  Your entry is not a legal number. It may include non-numeric characters.

- `<prefix>, history line not found`

  The command prefix you entered does not appear in the history buffer.

This command assigns an abbreviation to a commonly used command pathname. The format of this command is



INPUT PARAMETERS

abbreviation	A short term that will replace the command prefix. Whenever the abbreviation is entered, it is replaced with the command prefix. Then the entire command is executed.
command	A command (and optionally parameters) that replaces the abbreviation during execution.
parameters	Formal parameters, #0 to #9, that are replaced by actual parameters in the invocation line.

DESCRIPTION

The ALIAS command provides a shorthand method of entering data at the terminal. It assigns an abbreviation to a command prefix. Once the alias has been assigned, the CLI recognizes the abbreviation as if it were the entire command and executes the command. The alias stays in effect until you enter either a DEALIAS or a LOGOFF command. Alias expansion can be nested up to five times for ease of use (see example 5 below).

Entering ALIAS with no parameters causes the CLI to list all your previously defined aliases. However, if you have not previously defined any aliases, the CLI displays

```
USER ALIASES ARE:
```

Aliases are displayed page by page. If your list of aliases requires more than one screen, the CLI displays one screen followed by this message:

```
display more ? ([y] or n)
```

If you wish to see more, respond "Yes". Otherwise respond "No", and the CLI will return to the Human Interface.

You can create an alias for any command including the ALIAS command. The alias command prefix can be entered with or without formal parameters, which are replaced with actual parameters in the command invocation. Formal parameters should be preceded by a pound sign (#).

You can specify a maximum of ten formal parameters in a command prefix. Formal parameters are replaced in order. You cannot skip a parameter. This means that if you have three formal parameters and only two actual parameters, a null string will replace the third formal parameter, not the first or second. If you enter more actual parameters than formal parameters, the extra parameters are considered as another command parameter (see example 4 below). Separate actual parameters with blanks.

To assign an ALIAS for the first time, enter

It is also possible to change an existing alias. For example, if you have defined the alias a=mer, and now you wish to change it to a=merrr, you can enter

The CLI changes the alias and issues this message:

```
<abbreviation>, former alias removed
```

The following list shows some commonly used aliases. These may be helpful in defining your own set of aliases, however, they are not required.

ALIAS	COMMAND NAME
a	ALIAS
ad	:system:attachdevice
adf	:system:attachdevice wmf0 as :f:
aed	:lang:aedit
af	:system:attachfile
bk	BACKGROUND
crdir	:system:createdir
dd	system:detachdevice
df	:system:detachfile
h	HISTORY
install	submit :config:cmd/instal(wmf0)
logs	:system:logicalnames
ls	:system:dir $ sort
lpr	BACKGROUND(100,100) copy #0 to :lp:
mksys	submit :config:cmd/mksys(#0)
pmw	:system:permit #0 drau u=world
sh	:system:shutdown w=0

EXAMPLES

For the examples in this section, assume that these aliases have previously been assigned:

```
ALIAS PLM = :lang:plm286 #0.p28 noli
ALIAS S = :system:submit
ALIAS SU = super
```

1. To display all of your aliases, enter

   The CLI responds with a list of the aliases currently defined.

```
    USER ALIASES ARE:
PLM = :lang:plm286 #0.p28 noli
S = :system:submit
SU = super
```

   If you enter ALIAS and have not previously assigned any abbreviations, you receive the message

```
USER ALIASES ARE:
```

2. To see only one specific alias, enter

   The CLI responds with

```
PLM = :lang:plm286 #0.p28 noli
```

3. To see all of your aliases starting with the letter S, enter

   The response is

```
S = :system:submit
SU = super
```

4. To execute the PLM command displayed in example 2 and replace its parameter, enter

   The CLI replaces the formal parameter #0 with MINE and executes the command as though :LANG:PLM286 MINE.P28 NOLIST had been entered.

   If you enter

   the CLI executes

   ```
   :LANG:PLM286 MINE.P28 NOLIST PAGEWIDTH(132)
   ```

   adding PAGEWIDTH(132) as an additional command parameter. The CLI does not echo this command on the screen.

5. To use the nested alias feature, assume you have defined the following aliases.

   ```
   ALIAS PLM=:LANG:PLM286
   ALIAS PNL=PLM #0.P28 NOLIST
   ```

   Now when you enter PNL SOURCE, ALIAS first replaces "PNL" with "PLM #0.P28 NOLIST". It then assigns "SOURCE" to #0, and finally it replaces "PLM" with ":LANG:PLM286". The command that is executed is

   ```
   :LANG:PLM286 SOURCE.P28 NOLIST
   ```

ERROR MESSAGES

- `ALIAS, wrong alias syntax`

  The command syntax is not correct.

- `<parameter>, alias not found`

  The alias you entered is not in the list of declared aliases.

- `<parameter>, wild card is allowed only in the last character`

  You tried to list your aliases with a wild card, but the wild card was not the last character in the string.

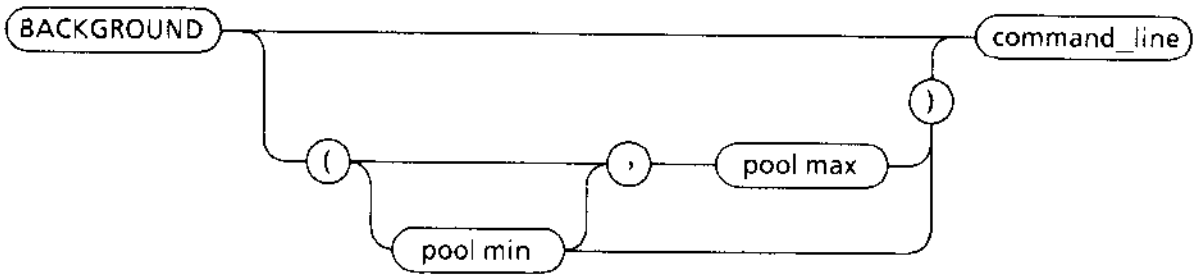
- `<parameter>, wild card not allowed in alias abbreviation`

  You declared an alias with a wild card. You can use wild cards only to display a list of aliases, not to define them.

- `ALIAS, no space in alias table`

  The alias table is full. No more aliases can be assigned unless you increase the size of the alias table or delete some less frequently used aliases. The default alias table size is 2K. You can use the SET command (described later in this chapter) to modify the size of the alias table.

This command causes the CLI to execute the command entered in a background environment. The format of this command is



F-0505

INPUT PARAMETERS

pool min	The minimum memory pool size to be allocated for background jobs. This value overrides the default pool min value defined using the SET command. The default pool min size is 6K.
pool max	The maximum memory pool size to be allocated for background jobs. This value overrides the default pool max value defined using the SET command. The default pool max size is calculated as either 300K or user$pool$max - 200K, whichever is smaller. (user$pool$max is defined during configuration of the terminal configuration file.) If pool max is less than 300K, the CLI sets pool max to zero.
command line	A user command to be executed in the background.

DESCRIPTION

When you enter this command, the CLI creates a background job to execute the command line. Background jobs are executed as they are submitted and are not queued. Each background job is assigned a four-digit hexadecimal job identification number (job ID) that you can display by entering the JOBS command. You can cancel background jobs by entering the KILL command. (JOBS and KILL are described later in this chapter.)

If the BACKGROUND command is invoked when the pool max value is zero, the CLI issues a message asking you to set the parameter with the SET command (described later in this chapter). The CLI then stops executing.

When the BACKGROUND command is entered, the active foreground environment is copied to the background job and becomes its initial environment. This means that the same logical names and aliases used in the foreground are also available to the background job. However, changes made to logical names and aliases in the background environment do not affect the foreground, and vice-versa. Each environment is independent of the other.

You can control the amount of memory allocated for the background jobs by entering the pool min and pool max parameters. The CLI checks that pool min is less than pool max. If pool max is less than pool min, the CLI issues this warning:

```
WARNING:    MAXBACKPOOL < MINBACKPOOL,
            use SET command to set BACKGROUND memory pools
```

If pool max is less than 300K, the CLI assigns a value of zero and issues this message:

```
MAXBACKPOOL attribute <300K, was set to 0
please SET your MAXBACKPOOL attribute
```

These modifiers are recommended for large programs such as compilers, ensuring the minimum memory pool for the application but leaving enough memory for other jobs (mainly in the foreground).

The BACKGROUND command will prompt for a redirect if you don't supply one. If you do not use the I/O redirection feature, the background process queries you for a log file to replace the terminal. The message displayed is

```
the log file is ?
```

The CLI expects you to respond with an iRMX II pathname. If you enter :CO: as the log file, the CLI displays the message

```
:CO:, not a valid log file
the log file is ?
```

However, if you have a system with multiple terminals, you can redirect :CI: and :CO: to another terminal that acts as a background terminal. A background job that tries to send a message to the operator's physical terminal (:EO:) causes the following message to appear on your screen:

```
***8085:  E$ERROR_OUTPUT
```

When the background job begins running, the CLI displays this message:

```
***CLI : background job <job id> "command" has been started
```

When the background job is complete, the CLI displays

```
***CLI : background job <job id> "command" completed
```

The command given in the above messages is always enclosed in quotation marks ("). The first 15 characters of the command are displayed.

EXAMPLES

1. This example illustrates using the BACKGROUND command and the I/O redirection feature to create a background job and send the output to a file named OUT.

```
BACKGROUND COPY X.ASM TO Y >OUT  <CR>
***CLI : background job <0168> "copy x.asm to y" has been started
```

When the background job is complete, the following message is displayed at the terminal:

```
***CLI : background job <0168> "copy x.asm to y" completed
```

The output file, OUT, contains all the output messages, such as

```
x.asm copied to y
```

2. This example shows the CLI prompt if you do not redirect your output:

```
BACKGROUND COPY X.ASM TO Y  <CR>
the log file is ?  OUT  <CR>
***CLI : background job <0E78> "copy x.asm to y" has been started
```

3. This example changes the default pool sizes of a background job by entering the pool min and pool max parameters (MINBACKPOOL, MAXBACKPOOL).

```
BACKGROUND (300,500) SUBMIT PLM >OUT   <CR>
***CLI : background job <0C68> "submit plm" has been started
```

ERROR MESSAGES

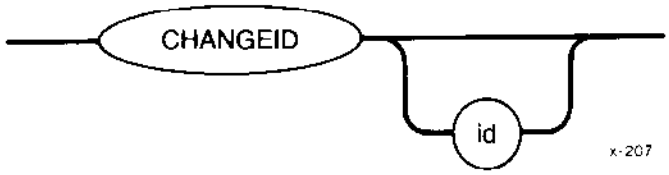
- `***CLI : background job <job id> "<command>" . failed`
  `***<error message>`

  The BACKGROUND command failed for the reason given in the error message.

- `BACKGROUND, parameter required`

  You entered the command without parameters.

This command can be issued only while in SUPER mode. With this command, a system manager can change your current user ID to any value between 0 and 65535 decimal. The format of the CHANGEID command is as follows:



INPUT PARAMETER

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

DESCRIPTION

With CHANGEID you can change your current user ID to any value between 0 and 65535 decimal after entering the SUPER command. If you change your user ID to anything other than that of the system manager (user ID 0), the system prompt changes to the following:

```
SUPER(id)-
```

where

id — Decimal equivalent of your new user ID (or the ASCII characters "WORLD" if the ID is 65535).

The new user ID created by entering CHANGEID is not a verified user. This means the new user cannot access the files available on the iRMX-NET network. You are not a verified user until you return to user ID 0.

ERROR MESSAGES

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

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

- `CHANGEID, allowed only in SUPER mode`

  You were not in SUPER mode when you invoked this command.

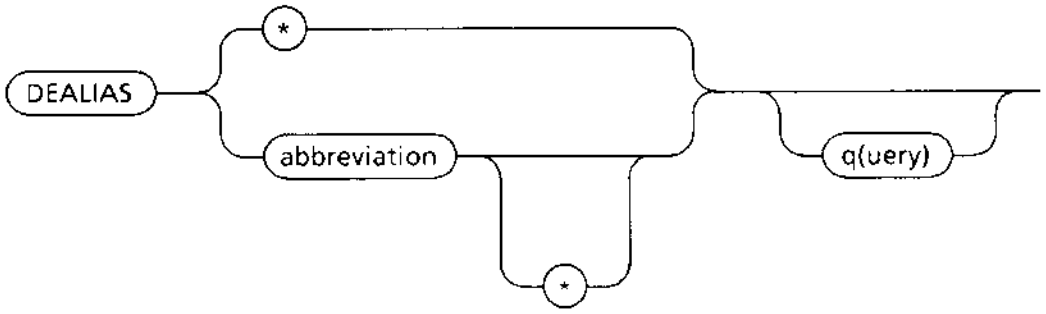
- `<parameter>, unexpected parameter`

  Too many parameters entered.

- `<exception value> : <exception mnemonic>, while executing CHANGEID`

  An internal system problem occurred which prevented the CLI from setting the default user.

This command deletes an alias. It has the following format:



INPUT PARAMETERS

abbreviation	The short term representing the alias abbreviation to be deleted.
query	Causes the CLI to prompt you for permission before deleting an alias.

DESCRIPTION

The DEALIAS command causes the CLI to delete an abbreviation established by the ALIAS command. If you enter the DEALIAS command with the query option, the CLI displays the alias and its command prefix followed by a question mark (?), then waits for your response before deleting the alias. NO is the default option. Entering anything other than Y(es), such as a carriage return, is equivalent to entering NO. If you wish to delete the alias, you must enter Y(es) or y(es).

EXAMPLE

This example illustrates the use of the query option. It demonstrates how to list all the aliases that start with the letter S and choose the one to delete. In this example, SU is deleted.

If you enter

The CLI displays

```
S = :system:submit delete ? (y or [n]) <CR>
SU = super delete ? (y or [n])  Y <CR>
```

ERROR MESSAGES

- `<parameter>, alias not found`

  You tried to delete an alias that was not defined in the alias table.

- `<parameter>, wild card is allowed only in the last character`

  You tried to delete a number of aliases with a wild card, but the wild card was not the last character.

This command exits from the SUPER mode. The format of this command is

DESCRIPTION

After you enter this command, the CLI changes your user ID back to the ID you had before entering the last SUPER command. It also changes the system prompt back to the prompt that was in effect before the SUPER command.

ERROR MESSAGES

- `EXIT, allowed only in SUPER mode`

  You invoked this command when you were not in SUPER mode.

- `<parameter>, unexpected parameter`

  You entered a parameter. EXIT does not require any parameters.

- `<exception value> : <exception mnemonic>, during EXIT execution`

  An internal system problem occurred which prevented the CLI from setting the default user.

This command displays the last 40 command lines on the screen. The format of this command is

F-0511

DESCRIPTION

The HISTORY command causes the CLI to display the last 40 command lines, including the HISTORY command, on the screen in chronological order. The command lines are displayed one screen at a time and are numbered from 1 to 999. After 999, the numbers wraparound. When displaying the command lines, the CLI lists the first page (20 lines) of command lines followed by the query

```
display more ? ([y] or n)
```

The default is y. If you enter anything other than n(o), the CLI displays the next page of command lines (assuming there are more command lines in the history buffer).

You can use the HISTORY command with the ! command to recall a specific line number or command line. For example, you might enter the HISTORY command to see the last 20 command lines. You could then recall line 10, modify it, and execute it. You would enter

This would display line 10 as the current line. You can then edit line 10. However, the original line 10 remains unchanged in the history. The edited line becomes the last line in the history record.

To recall a specific command, enter ! followed by the actual command as you have previously entered it. If you try to recall a line number or a command that cannot be found in the history buffer, the CLI issues one of the following messages:

```
<number>, history number not found
```

```
<prefix>, history line not found
```

EXAMPLE

A typical session at the terminal may look like this. The CLI echoes exactly what you enter. If your input is in lowercase, the display on the screen will be in lowercase. First, you might enter

The response would be

```
1 COPY X TO Y.PLM
2 DIR
3 AEDIT Y.PLM
4 HISTORY
```

Now suppose you decide to edit line 1. You can use the ! command and enter

The CLI displays

```
-COPY X TO Y.PLM
```

You can edit that line as follows:

This line is then entered into the history buffer as line number 5. Now, if you enter the HISTORY command, you will see:

```
1 COPY X TO Y.PLM
2 DIR
3 AEDIT Y.PLM
4 HISTORY
5 COPY NEW.PLM TO Y.PLM
6 HISTORY
```

If you have previously entered a series of command lines, in lowercase, that include a continuation line, entering the HISTORY command will cause the following screen display:

```
   1 copy x to y
   2 copy z &
 **  to &
 **  t.asm
   3 dir
   4 history
```

ERROR MESSAGES

- `<prefix>, history line not found`

  You tried to recall a command that cannot be found in the history buffer.

- `<number>, history number not found`

  You tried to recall a line number that cannot be found in the history buffer.

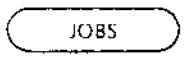
- `<parameter>, unexpected parameter`

  You entered a parameter. HISTORY does not accept parameters. If you want to recall a specific line, enter the ! command.

- `<exception value> : <exception mnemonic>, while history displayed`

  An error occurred when the CLI tried to write the history buffer to the screen.

This command displays all the existing background jobs by job identification number. The format of this command is



F 0512

DESCRIPTION

The JOBS command displays all the existing background jobs as a list of four-digit hexadecimal numbers. The jobs are displayed in reverse order, (last-in first-out) and are the job IDs assigned when the background jobs were invoked. If you wish to cancel a background job, enter the KILL command with a job ID displayed in the JOBS command.

The following example shows the use of this command:

```
JOBS <CR>
BACKGROUND JOBS ARE:
        1068            "<job>"
        1FF0            "<job>"
        10A8            "<job>"
```

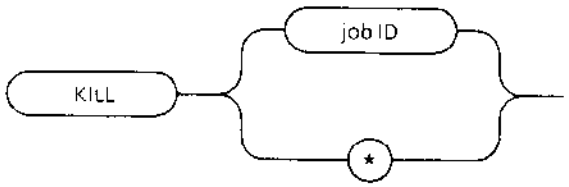
Where the term <job> is a truncated copy of the command line running in the background.

ERROR MESSAGE

- `<parameter>, unexpected parameter`

  You entered a parameter. JOBS does not accept any parameters.

This command cancels a specific background job. The format of this command is



F-0510

INPUT PARAMETER

job id	The job identification number established when the background job was invoked.
*	All background jobs.

DESCRIPTION

This command causes the CLI to cancel the specified background job(s). You can use the JOBS command to obtain a list of the jobs by ID number. Note, however, if you cancel several background jobs at once and then immediately issue the JOBS command, some of the canceled jobs may list at the console. Even though these jobs appear on the console, they have been canceled. To reassure yourself, issue another JOBS command. When the job has been canceled, you will receive the following message:

```
***CLI : background job <job id> canceled
```

If you use the * (asterisk) parameter when you invoke the KILL command, all background jobs will be canceled and you will receive the following message:

```
***CLI : all background jobs were canceled
```

ERROR MESSAGES

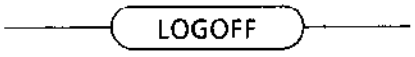
- `KILL, the job parameter is not a valid BACKGROUND job of the caller`

  You tried to kill a background job that is not in your list of background jobs.

- `KILL, a job parameter is required`

  The command you entered has a syntax error.

This command logs the user off of a dynamic logon terminal. The format of the command is as follows:



F-0664

DESCRIPTION

The LOGOFF command frees a dynamic logon terminal for use by other operators. It deletes the user's interactive job, executes the :PROG:R?LOGOFF file, and issues the logon prompt. On static logon terminals, LOGOFF simply terminates the session and restarts a new session for the same user.

If there are any active background jobs when the LOGOFF command is issued, you will receive the message

```
background jobs are running, do you want to exit ? ([n] or y)
```

By default, there are no logoffs while background jobs are active. If you respond with "Yes", your background jobs are canceled and you are logged off.

This command is also available as a Human Interface command, without background checking, for users who have a customized CLI.

ERROR MESSAGES

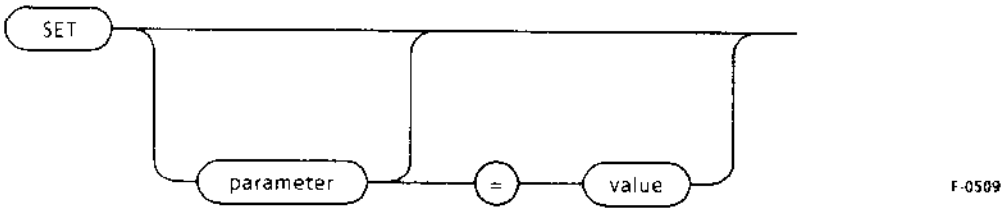
- `:prog:r?logoff, file does not exist`

  The CLI could not find the logoff file.

- `<parameter>, unexpected parameter`

  You have a syntax error. LOGOFF does not require any parameters.

This command alters the CLI environment by allowing on-line changes to the terminal name, the minimum and maximum memory pool sizes for background jobs, the size of the alias table, and the prompt. The format of this command is



INPUT PARAMETERS

parameter	One of the following keywords: TERMINAL, MINBACKPOOL, MAXBACKPOOL, ALIASTABLE, OR PROMPT.
value	The corresponding string or numerical value for each keyword.

DESCRIPTION

The SET command consists of five subcommands that alter the CLI environment.

- TERMINAL defines the terminal on which the CLI line-editing features will be supported.
- MINBACKPOOL and MAXBACKPOOL establish the minimum and maximum memory pool sizes for background jobs.
- ALIASTABLE sets the size of the alias table.
- PROMPT defines the CLI prompt.

If you enter this command with no parameters, the CLI displays the current values as follows:

```
set  <CR>
CLI PARAMETERS ARE:
        TERMINAL = ANY
        PROMPT = -
        MINBACKPOOL = 6K
        MAXBACKPOOL = 300K
        ALIAS TABLE SIZE = 2K
```

If you enter the command with an illegal parameter, the CLI prompts you as follows:

```
SET, illegal parameter, parameters are:
        TERMINAL        PROMPT          ALIASTABLE
        MINBACKPOOL     MAXBACKPOOL
```

If you enter the command with a parameter but without the = value, the CLI displays the existing value. For example, suppose you change the prompt from hyphen (-) to + + +, and then enter PROMPT with no parameters, the display would look like this:

```
- SET PROMPT = +++  <CR>
+++ SET PROMPT  <CR>
        PROMPT = +++
+++
```

SET TERMINAL

The SET TERMINAL command changes the current terminal definition to the new terminal name entered. (The initial terminal name is defined in the :CONFIG:TERMINALS file.) The format of the TERMINAL subcommand is

SET TERMINAL = <terminal name>

where <terminal name> is a string defining the terminal on which the CLI line-editing features are to be supported. The terminal must have been previously defined in the terminal definition file (see Appendix B for a sample definition file) or the CLI will issue the error message

```
<terminal name> is not found in :config;termcap
default ANSI standard assumed
```

After issuing this message, the CLI assumes the terminal name is the default ANSI standard until you redefine it using the SET command.

If you wish to add a terminal to the terminal definition file, you should edit the :CONFIG:TERMCAP file.

All assignments made with the SET TERMINAL command are valid for one logon session only. To change the terminal definition permanently, you must change the terminal configuration file, :CONFIG:TERMINALS (usually the system manager's responsibility).

SET MINBACKPOOL and SET MAXBACKPOOL

The format of these subcommands is

SET MINBACKPOOL=size
SET MAXBACKPOOL=size

MINBACKPOOL establishes the minimum memory pool to be allocated for background jobs. The default is 6K. This value can be overridden for a specific background job by entering the BACKGROUND command.

MAXBACKPOOL establishes the maximum memory pool to be allocated for background jobs. The default is 0, if this user has a maximum memory partition less than 500K. Otherwise, the default is 300K. This value can be overridden for a specific background job by entering the BACKGROUND command.

The values entered in these subcommands are decimal numbers that represent the amount of memory in Kbytes. However, the letter K should not be entered. For example, you should enter 1 for 1K bytes, 100 for 100K bytes etc.

These subcommands should be invoked with a minimum value large enough to accommodate the background stack and a maximum value less than (user$pool$max - 200K). If the maximum value is greater than (user$pool$max - 200K), you may not have enough memory to execute foreground jobs. Therefore, the CLI issues this warning:

```
WARNING:    MAXBACKPOOL attribute can avoid FOREGROUND execution
            due to memory limits
```

This is only a warning. The values are set as you enter them. However, you may want to re-examine the values you entered.

If the maximum value you enter is less than the minimum value, the CLI issues this warning:

```
WARNING:    MAXBACKPOOL < MINBACKPOOL, value was assigned
            use SET command to set BACKGROUND memory pools
```

The default values provide enough memory for most ordinary jobs.

SET ALIASTABLE

The ALIASTABLE subcommand establishes the size of the memory pool for the alias table. The size can be increased or decreased during a user session. The default is 2K. The format of this subcommand is

SET ALIASTABLE=size

The size value entered should be a decimal number representing the amount of memory in Kbytes. However, the letter K should not be entered. For example, you should enter 1 for 1K bytes, 5 for 5K bytes, etc.

SET PROMPT

The format of the PROMPT subcommand is

SET PROMPT=string

where string is a string of up to 14 characters that defines the CLI prompt. The default is "-".

ERROR MESSAGES

- `<terminal name>, is not found in :config:termcap:`
  `default ANSI standard assumed`

  The terminal name you entered is not defined in the terminal definition file. The default ANSI standard is assumed to be the terminal name until you redefine it using the SET command.

- `<alias size>, new alias table is not enough to hold user aliases`

  The new value you entered is too small to contain all the aliases you have assigned. The actual table size is not changed.

  This message does not appear if you reduce the size of the alias table and the new size is still large enough for all the aliases you have assigned. For example, if you reduce the alias table from 2K to 1K and all your aliases fit into 1K bytes of memory, no message is issued.

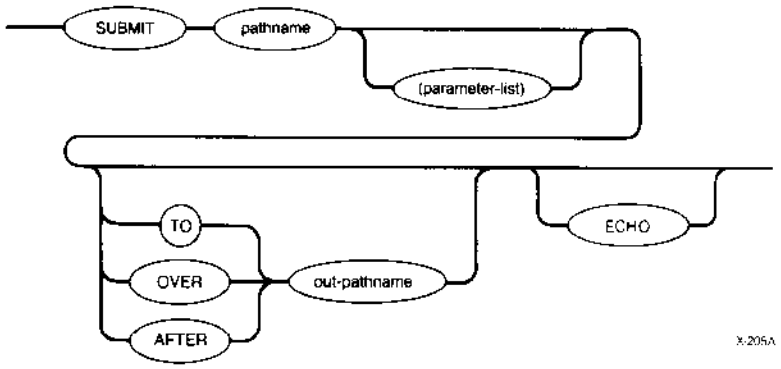
- `SET, wrong syntax`

  You have entered the command incorrectly.

- `SET illegal parameter, parameters are:`
  `TERMINAL PROMPT ALIASTABLE`
  `MINBACKPOOL MAXBACKPOOL`

  You entered the command with an illegal parameter.

This command reads and executes a set of commands from a file in secondary storage instead of from the console keyboard. The format of SUBMIT is



INPUT PARAMETERS

pathname — Name of the file from which the commands will be read. This file may contain nested SUBMIT commands.

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 ten parameters, separated by commas, in the SUBMIT command. If you omit a parameter, you must reserve its position by entering a comma. If a parameter contains a comma, space, or parenthesis, you must enclose the parameter in single quotes. The sum of all characters in the parameter list must not exceed 512 characters.

OUTPUT PARAMETERS

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

```
<pathname>, already exists OVERWRITE?
```

	Enter Y, y, R, or r if you wish the existing output file to be deleted. Enter any other character if you do not wish the existing file to be deleted. A response other than Y, y, R, or r causes the SUBMIT command to be terminated, and you will be prompted for a new command entry.
OVER	Causes the output for each command in the SUBMIT file to be written over the specified existing file instead of the console screen.
AFTER	Causes the output from each command in the SUBMIT file to be written to the end of an existing file instead of the console screen.
out-pathname	Pathname of the file to receive the processed output from each command executed from the SUBMIT file. If no preposition or output file is specified, TO :CO: is assumed.
ECHO	ECHO causes a copy of the data read from the first level of a SUBMIT file to be sent to the screen. This parameter lets you know which action specified within a SUBMIT file is currently executing. Nested SUBMIT commands do not have their contents sent to the console screen, unless they are invoked with their own ECHO.

DESCRIPTION

To use the SUBMIT command, you must first create a data file that defines the command sequence and formal parameters (if any). The operating system first looks for the pathname with the extension ".CSD". If no such file is found, then the operating system looks for the file specified in the pathname.

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

If, during the execution of SUBMIT (or any nested SUBMIT), you enter the CONTROL-C character to abort processing, all SUBMIT processing exits and control returns to you.

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

When you specify a preposition and output file (other than :CO:) in a SUBMIT command, only your SUBMIT command entry will be echoed on the console screen; the individual command entries in the submit file are not displayed on the screen as they are loaded and executed. However, if you specify the ECHO parameter in the same SUBMIT command, the command entries from the first level of the SUBMIT file will be sent to the screen. SUBMIT files invoked with nested SUBMIT commands will not have their contents sent to the screen unless they were invoked with their own ECHO parameter.

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

```
END SUBMIT <pathname>
```

You may use the CLI commands such as ALIAS and BACKGROUND within a SUBMIT file. In fact, the SUBMIT command is especially useful in background mode. You can use it to execute large tasks while you continue entering data from the terminal.

If you have a customized CLI or the Release 1 CLI of the Operating System, you can use the SUBMIT command as a Human Interface command. However, when using it as an HI command, you cannot take advantage of the CLI features such as aliasing and background.

EXAMPLE

This example shows how to use the SUBMIT command with the BACKGROUND command. It also shows how to replace formal parameters in the command that you enter to invoke this SUBMIT file. Assume the SUBMIT file, which resides in the file HLLCSP.CSD, contains the following lines:

```
BND286  &
    :F1:%0.OBJ, %2 %3  &
    /RMX286/HI/HUTIL.LIB,     &
    :LANG:PLM286.LIB,   &
    /RMX286/LIB/RMXIFC.LIB    &
RENAMESEG(HI_CODE TO CODE, HI_DATA TO DATA) NODB NOTY &
SEGSIZE (STACK(1200H)) OBJECT(:F1:%0) RC(DM(%1,0FFFFFH))
```

The SUBMIT command you use to invoke this file is as follows:

From this example you can see that the SUBMIT file contains four formal parameters. However, the invocation line has only two. This means that when the command is executed, COPY replaces %0 as the name of the object file, 5000 replaces %1 as the minimum dynamic memory requirement, and a null string replaces %2 and %3. After execution, the output file OUTFILE contains the following:

```
BND286   &
    :F1:HCOPY.OBJ,   &
    /RMX286/HI/HUTIL.LIB,   &
    :LANG:PLM286.LIB,   &
    /RMX286/LIB/RMXIFC.LIB   &
RENAMESEG(HI_CODE TO CODE, HI_DATA TO DATA) NODB NOTY &
SEGSIZE (STACK(1200H)) OBJECT(:F1:COPY) RC(DM(5000,0FFFFFH))
```

The execution of the above example takes place in the background environment, leaving you free to enter other data from the terminal. The CLI requires only a terminal to display a message that the job has started and that it has completed. When the background job starts, the CLI displays

```
***CLI : background job <1808> "  submit hllcsp" has been started
```

When the background job is complete, the CLI displays

```
***CLI : background job <1808> "  submit hllcsp" completed
```

Only the first 15 characters in the job name are displayed.

ERROR MESSAGES

- `<pathname>, end of file reached before end of command`

  The last command in the input file was not specified completely. For example, the last line might contain a continuation character.

- `<parameter>, incorrectly formed parameter`

  You separated the individual parameters in the parameter list with a separator character other than a comma.

- `<pathname>, output file same as input file`

  You attempted to place the output from SUBMIT into the input file.

- `<pathname>, too many input files`

  You specified more than one pathname as input to SUBMIT. SUBMIT can process only one file per invocation.

- `<parameter>, too many parameters`

  You specified more than ten parameters in your parameter list.

- `<pathname>, UPDATE or add access required`

  SUBMIT cannot write its output to the output file because you do not have update access to the file (if it already exists) or because you do not have add access to the file's parent directory (if the file does not exist).

- `<exception value> : <exception mnemonic>, during SUBMIT execution`

  The code in your SUBMIT file produced the error indicated.

With this command, you can change your user ID to the system manager ID (user ID 0). The format of this command is as follows:

DESCRIPTION

With SUPER you can change your user ID to that of the system manager. After entering SUPER, you can invoke the two related commands, CHANGEID and EXIT. CHANGEID changes your user ID to any valid value. EXIT exits the SUPER utility.

To invoke SUPER, you must know the password associated with the system manager. This password is stored in the file :CONFIG:UDF for user name SUPER (refer to the *Guide to the Extended iRMX II Interactive Configuration Utility* for more information). After you enter the SUPER command, SUPER prompts for the password by displaying

```
enter password:
```

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

```
SUPER-
```

This prompt is a system prompt (replacing the current prompt). It also indicates that you are now a verified user with the privileges of the system manager. Being a verified user allows you to access files via iRMX-NET. You can enter any commands and access any files available to the system manager. If you create new files, they will be listed as owned by user ID 0. You can also invoke the commands available with SUPER.

SUPER can be used only in the foreground. If you try to invoke it from a BACKGROUND command, you will receive the background failed message.

```
***CLI : background job <job-id> "SUPER" has been started
***CLI : background job <job-id> "SUPER" failed
***005: E$CONTEXT
```

This command is also available as a Human Interface command to users with a customized CLI. However, SUPER as an HI command does not recognize any of the CLI features such as line-editing and aliasing.

ERROR MESSAGES

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

  A problem prevented the CLI from changing your user ID. The UDF may be corrupted.

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

  An internal system problem occurred. For example, the CLI could not find the default user.

- `<exception value> : <exception mnemonic>, SUPER is unavailable`

  The CLI encountered an error while reading the password you entered or while accessing the user definition file (to determine if the password is correct).

- `<parameter>, unexpected parameter`

  You entered a parameter. The SUPER command does not require any parameters.

CHAPTER 4
HUMAN INTERFACE COMMANDS

4.1 OVERVIEW

This chapter presents the Human Interface commands in alphabetical order. The Human Interface Command Dictionary (Table 4-1) lists the commands in functional groups for quick reference.

These commands are supplied for Extended iRMX II Operating Systems configured with the Human Interface. If you are a new user of the Human Interface, you should review the information on file-naming conventions and invocation considerations in Chapter 2 before reading this chapter.

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

4.2 ERROR MESSAGES

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

- `command not found`

  There is no file with the pathname you specified, and the Human Interface cannot find the file in any of the directories it automatically searches.

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

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

- `<pathname>, file does not exist`

  The pathname specified does not represent an existing file.

- `<pathname>, invalid file type`

  A data file was specified for an operation that required a directory, or vice versa.

- `<logical name>, invalid logical name`

  The logical name specified contains unmatched colons, is longer than 12 characters, or contains invalid characters.

- `<pathname>, invalid pathname`

  The pathname specified contains invalid characters, or a component of the pathname (other than the last one) does not exist or does not represent a directory.

- `<logical name>, is not a device connection`

  The logical name specified does not represent a connection to a physical device.

- `<logical name>, logical name does not exist`

  The logical name specified does not exist.

- `parameters required`

  The command specified cannot be entered without parameters.

- `program version incompatible with system`

  The command and the operating system are not compatible. The command expects to obtain information from internal tables that are not present. Therefore the command cannot run successfully.

- `<control>, unrecognized control`

  The parameter entered is not valid for the specified command.

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

  The operating system encountered an exceptional condition while attempting to load the command into memory from secondary storage. The <exception value> and <exception mnemonic> portions of the message indicate the exception code encountered.

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

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

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

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

4.3 COMMAND SYNTAX SCHEMATICS

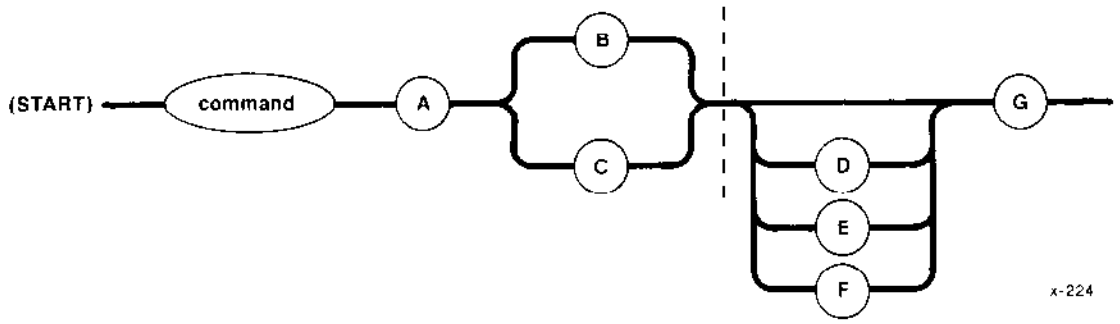
The syntax for each command described in this chapter is presented in a "railroad track" schematic, with syntactic elements scattered along the track. Your entrance to any schematic is always from left to right, beginning with the command name entry.

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. Syntactic elements shown in lowercase characters are generic terms, which means that you supply the specific item, such as the pathname for a file.

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

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

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



In this example

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

Table 4-1. Human Interface Command Dictionary

Command	Function	Page
	File Management Commands	
ATTACHFILE	Associates a logical name with an existing file.	4-26
COPY	Copies files specified in an input list to files specified in an output list.	4-38
CREATEDIR	Creates one or more new directories.	4-42
DELETE	Deletes data files and empty directories from a secondary storage device.	4-51
DETACHFILE	Removes the association of a logical name with a file.	4-55
DIR	Lists a directory's filenames (and, optionally, file attributes).	4-57
DOWNCOPY	Copies files from an Extended iRMX II volume mounted on a secondary storage device to an Intellec Development System secondary storage device via the iSDM monitor.	4-74
PERMIT	Grants or rescinds user access to a file.	4-126
RENAME	Changes the names of files or directories.	4-132
UPCOPY	Copies files, via the iSDM monitor, from an Intellec Development System secondary storage device to an Extended iRMX II volume mounted on a secondary storage device.	4-157

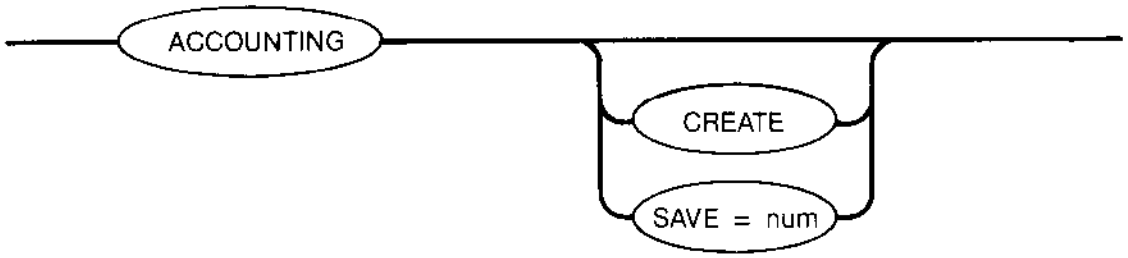
Table 4-1. Human Interface Command Dictionary (continued)

Command	Function	Page
	Volume Management Commands	
ATTACHDEVICE	Attaches a new physical device to the system and catalogs its logical name in the root job's object directory.	4-16
BACKUP	Copies named files to a backup volume.	4-29
DETACHDEVICE	Removes a physical device from system use and deletes its logical name from the root job's object directory.	4-53
DISKVERIFY	Verifies the data structures of named and physical volumes.	4-66
FORMAT	Writes format information on an iRMX II volume.	4-77
RESTORE	Copies files from a backup volume to a named volume.	4-135
	System Management Commands	
ACCOUNTING	Tracks activities of dynamic logon users.	4-8
INITSTATUS	Displays the initialization status of Human Interface terminals.	4-95
JOBDELETE	Deletes a running interactive job.	4-98
LOCK	Prevents the Human Interface from automatically creating an interactive job.	4-105
LOGOFF	Ends a user session for users with a customized CLI.	4-111
PASSWORD	Changes passwords for dynamic logon users and creates new users when invoked by the system manager.	4-113
SUPER	Changes the operator's user ID into that of the system manager (user ID 0) for users who are using a custom CLI.	4-151
UNLOCK	Permits the Human Interface to create an interactive job, after the terminal has been locked by the LOCK command.	4-155

Table 4-1. Human Interface Command Dictionary (continued)

Command	Function	Page
	General Utility Commands	
ADDLOC	Combines the output of LOCDATA and an iRMX II bootloadable file. The output of ADDLOC is another iRMX II bootloadable file.	4-12
DATE	Sets or resets the system date, or displays the current date.	4-44
DEBUG	Transfers control to the iSDM monitor to debug an iRMX II application program.	4-48
LOCDATA	Reads the specified data and creates a "located" file that can be processed by the ADDLOC command.	4-100
LOGICALNAMES	Lists all the logical names available to the user.	4-107
MEMORY	Displays the memory available to the user.	4-112
PATH	Shows the pathname for a file.	4-123
PAUSE	Displays optional message to the console.	4-125
RETENSION	Retensions a tape.	4-143
SHUTDOWN	Provides orderly shutdown of the system.	4-144
SUBMIT	Reads, loads, and executes a string of commands from secondary storage instead of from the keyboard for users with a custom CLI.	4-150
TIME	Sets or resets the system clock, or displays the current system time.	4-152
VERSION	Displays the version numbers of commands.	4-160
WHOAMI	Displays the current ID associated with the user.	4-162
ZSCAN	Lists the ZAPs (updates) applied to an object module, library, or bootloadable file.	4-163

ACCOUNTING creates, lists the contents of, or reduces the size of the file :CONFIG:ACCOUNT.LOG. This file contains the logon and logoff history of dynamic logon users. The format of the command is as follows:



x-1113

INPUT PARAMETERS

CREATE — Creates a new accounting file to store the logon and logoff history. You must be the system manager to use this parameter. If :CONFIG:ACCOUNT.LOG already exists, ACCOUNTING displays the following message:

```
:config:account.log, already exists, OVERWRITE?
```

Enter Y, y, R, or r to delete the old accounting file and create a new empty file. Enter any other character to leave the old accounting file intact.

SAVE — Reduces the size of the accounting file by saving only the most recent num event entries, where num is a decimal number. All earlier entries are deleted from the file. You must be the system manager to use this parameter.

DESCRIPTION

ACCOUNTING enables you to observe the logon and logoff activities of all dynamic users in your application system. It also enables you to create and modify the file that stores this information.

For the ACCOUNTING command to be effective, the system manager must first use the CREATE parameter to create an empty :CONFIG:ACCOUNT.LOG accounting file. The CREATE parameter places special information in the file. Therefore, you must use this method, not a text editor, to create the :CONFIG:ACCOUNT.LOG file.

Once :CONFIG:ACCOUNT.LOG exists, the Human Interface then records all logon and logoff activities in that file. When you invoke the ACCOUNTING command with no parameters, the command lists the logon and logoff activities, beginning with the most recent activity. Figure 4-1 illustrates the format of the ACCOUNTING display.

```
 USER  USER      TERMINAL
   ID  NAME      DEVICE NAME  DATE        TIME      EVENT

    0  bob       .t2.         13 AUG 86   16:22:50  logoff
world  newuser   .t1.         13 AUG 86   14:45:00  logoff
world  newuser   .t1.         13 AUG 86   13:01:10  logon
    0  bob       .t2.         13 AUG 86   11:05:45  logon
    0  bob       .t2.         13 AUG 86   11:05:15  logon error 004B
```

Figure 4-1. ACCOUNTING Display

The columns listed in Figure 4-1 contain the following information:

USER ID	User ID associated with the user who engaged in a logon or logoff activity at a dynamic logon terminal.
USER NAME	Logon name used in the logon or logoff attempt.
TERMINAL DEVICE NAME	Physical name of the terminal, as defined during the configuration of the Basic I/O System and as attached by the Human Interface. Periods surround each name.
DATE	Date of the logon or logoff activity.
TIME	Time of the logon or logoff activity.

EVENT	Event that occurred. Three types of events can be listed:

logon	Indicates a successful logon.
logon error	Indicates an unsuccessful logon attempt. The resulting exception code is also listed.
logoff	Indicates that the user logged off the terminal.
logoff job deleted	Indicates that the JOBDELETE command was used to terminate a job. Note that the SHUTDOWN command can also cause this event to occur.
logoff carrier lost	Indicates that a terminal connected to a modem lost the carrier.

The system manager can use this command to determine system usage.

If the :CONFIG:ACCOUNT.LOG file grows too large to be manageable, or if some of the earlier information is not needed, the system manager can use the SAVE parameter to save only the most recent information. When the system manager invokes the ACCOUNTING command with the SAVE parameter, the command displays the following message to indicate the number of events saved and the number discarded:

```
<n> events saved; <m> events deleted
```

In the actual message, <n> and <m> are decimal numbers. The command then lists the events still recorded in the accounting file.

To stop the operating system from keeping track of logon and logoff activity, delete or rename the file :CONFIG:ACCOUNT.LOG.

ERROR MESSAGES

- `<exception-code>: <exception-name>, ACCOUNT.LOG is not available`

  The file ACCOUNT.LOG exists but is not currently available for reading or writing. The ACCOUNTING command terminates when this occurs.

- `:config:account.log, file does not exist`

  The accounting file for dynamic logon users does not exist.

- `not a valid accounting log file`

  The file :CONFIG:ACCOUNT.LOG exists, but it is corrupted, it doesn't contain accounting information, or it wasn't created with the CREATE parameter. Use the ACCOUNTING command with the CREATE parameter to create a new :CONFIG:ACCOUNT.LOG file.

- `only the system manager may change the accounting log file`

  Someone other than the system manager attempted to use the ACCOUNTING command with the CREATE or SAVE parameters.

- `program version incompatible with accounting log file`

  The :CONFIG:ACCOUNT.LOG file contains accounting information but is incompatible with this version of the ACCOUNTING command.

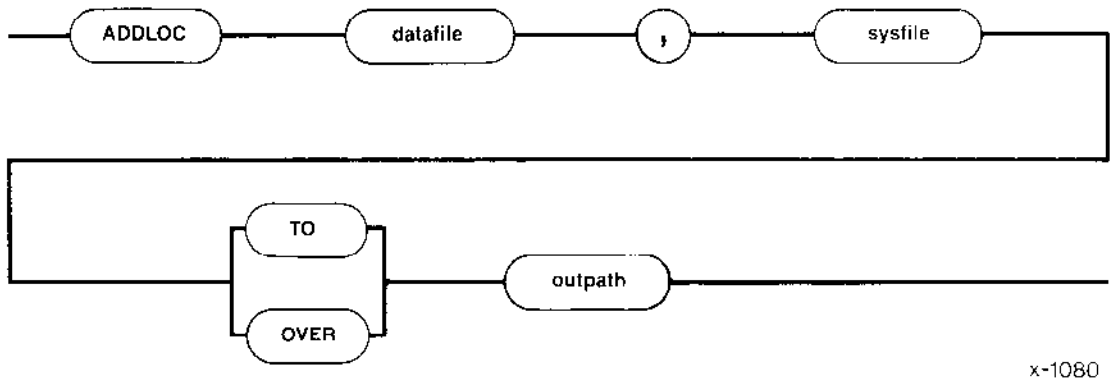
- `<exception-code>: <exception-name>, while attaching accounting log file`

  ACCOUNTING encountered an exceptional condition while attaching the existing accounting file. This message lists the resulting exception code.

- `<exception-code>: <exception-name>, while creating accounting log file`

  ACCOUNTING encountered an exceptional condition while creating a new accounting file. This message lists the resulting exception code.

ADDLOC, along with the Human Interface command LOCDATA, integrates the image of a data stream file (such as a RAM disk) into an existing application system boot file. ADDLOC adds the output of the LOCDATA command to a bootable application system. The result is an application system boot file that includes the data stream file (usually the RAM disk). ADDLOC also creates a map file that provides information about the new bootloadable file and the process that created it. The format of this command is as follows:



INPUT PARAMETERS

datafile — Pathname of the located data file (the output of the LOCDATA command) to be added to the bootloadable application system file. Multiple or wild-card pathnames are not allowed.

sysfile — Pathname of an object module format (OMF286) bootloadable application system file, to which the located data file is added. This file must have been created by the iAPX 286 SYSTEM BUILDER (invoked by the iRMX II ICU system generation file). Multiple or wild-card pathnames are not allowed.

OUTPUT PARAMETERS

TO — Writes the processed output to a named file. If the specified file already exists, ADDLOC displays the following message:

```
<pathname>, already exists, OVERWRITE?
```

To overwrite the existing file, enter Y, y, R, or r. If you do not wish to overwrite the existing file, enter E, e, N, or n. ADDLOC then exits without processing the data.

OVER	Overwrites the existing output file. If the specified file does not already exist, ADDLOC creates it.
outpath	The pathname of the file (up to ten characters) to receive the output of ADDLOC (the new OMF286 bootloadable file). This same pathname with the extension ".MPA" is also the print file of the process. Multiple or wild-card pathnames are not allowed.

DESCRIPTION

The Extended iRMX II Operating System supports the use of a RAM disk, an area of memory that is treated as a secondary storage device. To use the RAM disk feature you must configure a system with an area of RAM dedicated to the RAM disk. When the system boots, you can attach the RAM disk memory to your system, format it, and move data into and out of it just as you would with any other secondary storage device. If you use the RAM disk to store part of the application system (for instance, the Human Interface CUSPs), the stored data must be available in the RAM disk area when the system boots. This data cannot be copied into the RAM disk until you have configured the application system into a bootable file. (The RAM disk area does not exist until you define it through the configuration process.) Therefore, you must integrate a copy of a RAM disk data structure into an existing application system boot file.

ADDLOC uses the application system bootfile and the file output by LOCDATA (from the RAM disk) to create a file containing a new bootloadable version of the application system. The new version includes a copy of the RAM disk data structure. When this new file is booted, the RAM disk data structure is loaded into memory in the area defined for the RAM disk through the configuration process. The Human Interface LOCDATA command (later in this chapter) describes this process. (Users familiar with the iRMX I Operating System may remember that this mechanism was provided by the Human Interface LOCDATA command and the LIB86 utility in the iRMX I Operating System. LOCDATA transformed an image of the RAM disk into a module that LIB86 could add to the library containing the bootloadable application system.)

When you invoke ADDLOC, the first parameter you supply must be the pathname of the RAM disk datafile output by LOCDATA; the second parameter must be the pathname of the bootloadable application system file created by the iAPX 286 Builder; the third parameter must specify the pathname to be assigned to the new version of the application system. If the first parameter is a file that has not been processed by LOCDATA or if the second parameter is a file that has not been created by BLD286, ADDLOC will issue the following error message:

```
usage:  ADDLOC <located data file>, <system file> TO/OVER <outpath>
```

and exit without processing the data.

When processing has successfully completed, ADDLOC displays one of the following messages:

```
<located data file> added to <system file> TO <outpath>
```

```
<located data file> added to <system file> OVER <outpath>
```

In addition to the new iRMX II bootloadable file, ADDLOC also creates a print file. The print file is named by adding the extension ".MPA" to the name of the bootloadable output file. (Thus, if the bootloadable file output by ADDLOC is named "NEWSYS.286", the print file is named "NEWSYS.MPA") The print file contains a header that provides the name of the input and output files, the address space used by both the system file and the located data file, and the base address of the located data file. Following the header is a list of any error messages ADDLOC may have generated.

ERROR MESSAGES

- `ADDLOC, two input files only`

  ADDLOC requires exactly two input files. You specified more (or less) than two.

- `ADDLOC, one output file only`

  ADDLOC requires exactly one output file. You specified more than one.

- `ADDLOC, missing parameters`

  In your invocation line you failed to enter one or more required parameters.

- `AFTER, is an illegal preposition for ADDLOC`

  The AFTER preposition, which you entered in your invocation line, is not a legal ADDLOC preposition.

- `<string>, illegal preposition`

  The preposition entered in your invocation line is not a legal ADDLOC preposition.

- `<pathname>, output file same as input file`

  You specified that the input file name also be used as the output file name. ADDLOC does not allow this.

- `<pathname>, print file same as output file`

  The output file name you specified has the same name as the print file. (The .MPA extension denotes a print file.)

- `<pathname>, output pathname too long`

  The name of the file you specified in the output pathname exceeded ten characters.

- `<string>, unrecognized control`

  You entered an unrecognized control character in the invocation line.

- `<pathname>, write error`

  A system error caused an incorrect number of bytes to be written to the output file. Retry the command.

- `<pathname>, read error`

  A system error caused an incorrect number of bytes to be read from the input file. Retry the command.

- `<pathname>, not a located data file`

  The file was not processed by LOCDATA.

- `<pathname>, file does not exist`

  The file does not exist.

- `<pathname>, not a bootloadable file`

  The system file was not a system image file.

In addition to the error messages listed above, ADDLOC produces the three warning messages listed below. After each message, the ADDLOC command lists the file that caused the warning, the physical address, and the length of the section containing the faulty parameter.

- `OVERLAPPING AREAS IN MEMORY`

  The section read from the system file overlaps memory that was assigned to the located data stream. Although the process continues, the output is invalid.

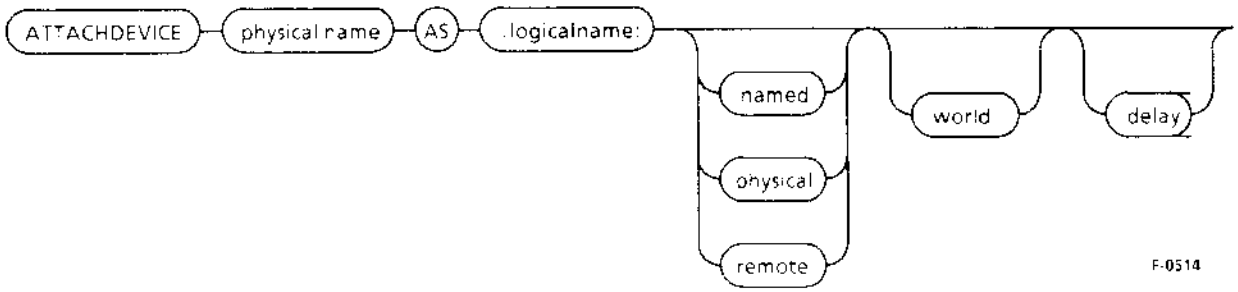
- `BAD SEQUENCE`

  The located data file contains a section that is not contiguous to the previous section. Although the process continues, the output is invalid.

- `BAD CHECKSUM`

  One of the input files you specified has a bad checksum. Output is invalid.

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



INPUT PARAMETERS

physical name	Physical device name of the device to be attached to the system. This name must be the name used in one of the Basic I/O System's Device Unit Information Blocks (DUIB), as defined at system configuration time (see Table 4-2).
AS	Preposition; required for the command.
logical name	The name (1 to 10 characters, or 1 to 12 characters if colons are included) to be associated with the device. Colons surrounding the logical name are optional; however, if you use them, you must use them in pairs.
NAMED	Specifies that the volume mounted on the device is already formatted for NAMED files. Examples of volumes that can contain named files are diskettes or hard disk platters. If NAMED, PHYSICAL or REMOTE are not specified, NAMED is the default. See the FORMAT command in this chapter for a further description of NAMED volumes.
PHYSICAL	Specifies that the volume mounted on the logical device is considered to be a single, large file. Examples include line printers, terminals, and tape drives. See the FORMAT command in this chapter for a further description of PHYSICAL volumes.
REMOTE	Specifies that the volume mounted on the logical device is a remote file server. If this parameter is selected, a logical name is created for the public system directory of the file server. The logical name can then be used to access files residing at the file server without any concern as to whether the file is local or remote. See the *iRMX Networking Software User's Guide* for more information on REMOTE volumes.

WORLD	Specifies that user ID WORLD (65535 decimal) is the owner of the device. This implies that any user can detach the device. If you omit this parameter, your user ID is listed as the owner of the device. In this case, only you and the system manager can detach the device.
DELAY	Use with named volumes. Causes the device to be attached logically as a named volume, but postpones the reading of the volume label until the first access.

DESCRIPTION

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

Table 4-2 lists the physical device names normally used with the Basic I/O System. Your system might support a subset of these devices or it might support devices not listed. If it supports the devices listed, it might support them under different names. Therefore, consult the person who configured your system to determine the correct device names for your system.

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

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

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

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

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

If you try to attach a device that has not been shutdown properly, you will receive this message:

```
<logical-name>, device was not shut down properly
```

where <logical-name> is the logical name of the device you are trying to attach.

Table 4-2 lists the physical device names supplied in the standard definition files. If you purchased a System 300 Series Microcomputer and are formatting the hard disk drive, refer to the *Extended iRMX II Hardware and Software Installation Manual* for the Feature Code or Option Code describing the exact hard disk drive in your microcomputer. All standard definition files for System 300 Series Microcomputers define the hard disk drives with the same device names.

Table 4-2. Physical Device Names in the Standard Definition Files For Devices Controlled by the iSBC 214/215G/iSBX 217C/218A Controller

Physical Device Names	Device Type	Unit Number	Sides	Density	Bytes per Sector	Tracks per Inch
		5.25-Inch Diskette Drives				
WMF0	Shugart 450	8	2	Double	512	48
WMF1	Shugart 450	9	2	Double	512	48
WMFDY0	Shugart 460	8	2	Double	512	96
WMFDY1	Shugart 460	9	2	Double	512	96
		8-Inch Diskette Drives				
WF0	Shugart SA800	8	1	Single	128	77
WF1	Shugart SA800	9	1	Single	128	77
WFD0	Shugart SA800	8	1	Double	256	77
WFD1	Shugart SA800	9	1	Double	256	77
WFDD0	Shugart SA850/SA851	8	2	Double	256	77
WFDD1	Shugart SA850/SA851	9	2	Double	256	77
WFDX0	Shugart SA850/SA851	8	2	Double	1024	77
WFDX1	Shugart SA850/SA851	9	2	Double	1024	77

Physical Device Names	Device Type	Unit Number	Bytes per Sector
	Hard Disk Drives		
W0	generic drive	0	1024
W1	generic drive	1	1024
IW0	Priam 3450 (8")	0	1024
CM0	CMI 5412	0	1024
CM1	CMI 5412	1	1024
CMB0	CMI 5419 and Fujitsu M2235	0	1024
CMB1	CMI 5419 and Fujitsu M2235	1	1024
MMA0	Maxtor XT-1140	0	1024
MMA1	Maxtor XT-1140	1	1024
MMB0	Maxtor XT-1085	0	1024
MMB1	Maxtor XT-1085	1	1024
QMA0	Quantum Q540	0	1024
QMA1	Quantum Q540	1	1024
TMA0	Toshiba MK56FB	0	1024
TMA1	Toshiba MK56FB	0	1024
	5.25-Inch Cartridge Tape Drives		
WTA0	Archive	12	N/A

Table 4-3 lists the physical device names of the terminals supported in the standard definition files. Unit/Board refers to the unit number and board number of the terminal controller. Where no board number appears, only one board has been selected in the ICU definition file. Where a board number appears, the ICU definition file selects multiple reincarnations of the controller board.

Table 4-3. Physical Device Names in the Standard Definition Files For Terminals

Device Names	Terminal Controller	Unit/Board Number	Supplied in the following Standard Definition File(s)
T0	8251A	0	38620.DEF, SXM386.DEF
T0	8274 Ch B	1	28612.DEF
T1	8274 Ch A	0	28612.DEF
T0	82530 Ch B	1	286100A.DEF
T1	82530 Ch A	0	286100A.DEF
T0	82530 Ch A	0	386100.DEF
T1	82530 Ch B	1	386100.DEF
T2	SBC 544A	0	28612.DEF, SXM386.DEF
T3	SBC 544A	1	28612.DEF, SXM386.DEF
T4	SBC 544A	2	28612.DEF, SXM386.DEF
T5	SBC 544A	3	28612.DEF, SXM386.DEF
T48_0	iSBC 188/56(48)	0	28612.DEF, 38620.DEF, SXM386.DEF
T48_1	iSBC 188/56(48)	1	28612.DEF, 38620.DEF, SXM386.DEF
T48_2	iSBC 188/56(48)	2	28612.DEF, 38620.DEF, SXM386.DEF
T48_3	iSBC 188/56(48)	3	28612.DEF, 38620.DEF, SXM386.DEF
T48_4	iSBC 188/56(48)	4	28612.DEF, 38620.DEF, SXM386.DEF
T48_5	iSBC 188/56(48)	5	28612.DEF, 38620.DEF, SXM386.DEF
T48_6	iSBC 188/56(48)	6	28612.DEF, 38620.DEF, SXM386.DEF
T48_7	iSBC 188/56(48)	7	28612.DEF, 38620.DEF, SXM386.DEF
T410_0	iSBC 186/410	0	286100A.DEF, 386100.DEF
T410_1	iSBC 186/410	1	286100A.DEF, 386100.DEF
T410_2	iSBC 186/410	2	286100A.DEF, 386100.DEF
T410_3	iSBC 186/410	3	286100A.DEF, 386100.DEF
T410_4	iSBC 186/410	4	286100A.DEF, 386100.DEF
T410_5	iSBC 186/410	5	286100A.DEF, 386100.DEF
T548_0	iSBC 548	0	28612.DEF, SXM386.DEF
T548_1	iSBC 548	1	28612.DEF, SXM386.DEF
T548_2	iSBC 548	2	28612.DEF, SXM386.DEF
T548_3	iSBC 548	3	28612.DEF, SXM386.DEF
T548_4	iSBC 548	4	28612.DEF, SXM386.DEF
T548_5	iSBC 548	5	28612.DEF, SXM386.DEF
T548_6	iSBC 548	6	28612.DEF, SXM386.DEF
T548_7	iSBC 548	7	28612.DEF, SXM386.DEF

Table 4-3. Physical Device Names in the Standard Definition Files For Terminals (continued)

Device Names	Terminal Controller	Unit/Board Number	Supplied in the following Standard Definition File(s)
T546_0	iSBC 546	0	38620.DEF
T546_1	iSBC 546	1	38620.DEF
T546_2	iSBC 546	2	38620.DEF
T546_3	iSBC 546	3	38620.DEF
T547_0	iSBC 547	0/1	38620.DEF
T547_1	iSBC 547	1/1	38620.DEF
T547_2	iSBC 547	2/1	38620.DEF
T547_3	iSBC 547	3/1	38620.DEF
T547_4	iSBC 547	4/1	38620.DEF
T547_5	iSBC 547	5/1	38620.DEF
T547_6	iSBC 547	6/1	38620.DEF
T547_7	iSBC 547	7/1	38620.DEF
T547_8	iSBC 547	0/2	38620.DEF
T547_9	iSBC 547	1/2	38620.DEF
T547_10	iSBC 547	2/2	38620.DEF
T547_11	iSBC 547	3/2	38620.DEF
T547_12	iSBC 547	4/2	38620.DEF
T547_13	iSBC 547	5/2	38620.DEF
T547_14	iSBC 547	6/2	38620.DEF
T547_15	iSBC 547	7/2	38620.DEF
T547_16	iSBC 547	0/3	38620.DEF
T547_17	iSBC 547	1/3	38620.DEF
T547_18	iSBC 547	2/3	38620.DEF
T547_19	iSBC 547	3/3	38620.DEF
T547_20	iSBC 547	4/3	38620.DEF
T547_21	iSBC 547	5/3	38620.DEF
T547_22	iSBC 547	6/3	38620.DEF
T547_23	iSBC 547	7/3	38620.DEF

Table 4-4 lists suggested physical devices names for other devices.

Table 4-4. Suggested Physical Device Names For Other Devices

Physical Device Names	Device Type	Unit Number	Sides	Density	Bytes per Sector	Tracks per Inch
8-inch Diskette Drives Controlled by the iSBC 208 Board						
AF0	Shugart SA800	0	1	Single	128	77
AF1	Shugart SA800	1	1	Single	128	77
AFD0	Shugart SA800	0	1	Double	256	77
AFD1	Shugart SA800	1	1	Double	256	77
AFDD0	Shugart SA850/SA851	0	2	Double	256	77
AFDD1	Shugart SA850/SA851	1	2	Double	256	77
AFDX0	Shugart SA850/SA851	0	2	Double	1024	77
AFDX1	Shugart SA850/SA851	1	2	Double	1024	77
5.25-inch Diskette Drives Controlled by the iSBC 208 Board						
AMF0	Shugart 450	0	2	Double	512	48
AMF1	Shugart 450	1	2	Double	512	48
AMFDY0	Shugart 460	0	2	Double	512	96
AMFDY1	Shugart 460	1	2	Double	512	96
5.25-inch Diskette Drives Controlled by the iSBX 218A Board When Mounted on a Processor Board						
PMF0	Shugart 450	0	2	Double	512	48
PMF1	Shugart 450	1	2	Double	512	48
5.25-inch Diskette Drives Controlled by the iSBC 186/224A Board						
WF0	This controller	4	2	Double	128	48
WF1	supports any	5	2	Double	128	48
WMF0	double-sided,	4	2	Double	512	48
WMF1	double-density	5	2	Double	512	48
WMFDY0	flexible disk	4	2	Double	512	96
WMFDY1	drive	5	2	Double	512	96

Table 4-4. Suggested Physical Device Names For Other Devices (continued)

Physical Device Names	Device Type	Unit Number	Sides	Density	Bytes per Sector	Tracks per Inch
Winchester Disk Drives Controlled by the iSBC 186/224A Board						
W0	generic	0			1024	
W1	generic	1			1024	
CM0	CMI 5412	0			1024	
CM1	CMI 5412	1			1024	
CMB0	CMI 5419 and Fujitsu M2235	0			1024	
CMB1	CMI 5419 and Fujitsu M2235	1			1024	
QMA0	Quantum Q540	0			1024	
QMA1	Quantum Q540	1			1024	
MMA0	Maxtor XT-1140	0			1024	
MMA1	Maxtor XT-1140	1			1024	
MMB0	Maxtor XT-1085	0			1024	
MMB1	Maxtor XT-1085	1			1024	
5.25-Inch Cartridge Tape Drives Controlled by the iSBC 186/224A Board						
WTA0	QIC-02	12			N/A	
Storage Module Disk (SMD) Drives Controlled by the iSBC 220 Board						
SMD0		0			1024	
SMD1		1			1024	
Bubble Memory Devices Controlled by the iSBX 251						
BX0		0			256	
Bubble Memory Devices Controlled by the iSBX 264						
BA0		0			256	

Table 4-5 lists SASI/SCSI controllers connected to an iSBC® 286/100A or iSBC® 386/100 host.

Table 4-5. SASI/SCSI Controllers Connected to an iSBC® 286/100A or iSBC® 386/100 Host

Device Names	Controller	Drive	Unit Number	Bytes per Sector
S0	Generic	any	0	512
SAT0	Adaptec ACB-4000	CMI 5619	0	512
SX1410A0*	Xebec 1410	CMI 5619	0	512
SX1410B0*	Xebec 1410	Quantum Q540	0	512
SX1420A0*	Xebec 1420	CMI 5619 and Fujitsu M2235	0	512
SX1420B0*	Xebec 1420	Quantum Q540	0	512
SX1420C0*	Xebec 1420	Maxtor XT-1140	0	512
SMF0	Xebec 1420	Teac F55B	2	512
SMF1	Xebec 1420	Teac F55B	3	512

* These controllers support the ST506 Winchester Interface

ERROR MESSAGES

- `<physical name>, cannot attach device`

  There is a hardware problem.

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

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

- `<physical name>, device already attached`

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

- `<physical name>, device does not exist`

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

- `<logical name>, logical name already exists`

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

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

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

- `<logical name>, device was not shut down properly`

  The device you attached was not previously shutdown using either the SHUTDOWN or the DETACHDEVICE command.

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

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

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

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

- `<logical name>, volume not mounted`

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

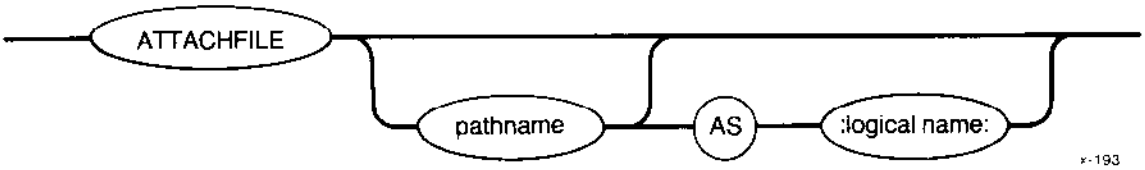
- `<exception value> : <exception mnemonic>, while collecting device name`

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

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

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

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



INPUT PARAMETERS

pathname	Pathname of the file to which the Human Interface associates a logical name.
:logical name:	The name (1 to 10 characters, or 1 to 12 characters if you include the colons) that represents the logical name to be associated with the file. Colons surrounding the logical name are optional; however, if you use them, you must use them in pairs. If you omit this parameter, the default logical name is :$:.

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

ATTACHFILE :HOME: AS :$:

DESCRIPTION

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

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

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

where <pathname> and <logical name> are as specified above.

ATTACHFILE makes the association between a file and a logical name by cataloging a connection to the file in your global object directory (this is normally the object directory of your interactive job). It catalogs the connection under the name specified as the logical name. If there is another connection cataloged in the object directory under the same logical name, ATTACHFILE uncatalogs and deletes the previous connection before cataloging the new one. If an object other than a connection is cataloged in the directory under the specified logical name, ATTACHFILE leaves the previous object as is, does not catalog the new connection, and displays an error message to describe the situation.

Because ATTACHFILE catalogs the connection in your global object directory, the logical name has effect only within your interactive job. Therefore, several users can specify the same logical name without affecting the others. Background jobs can also attach files without affecting the tasks being run in the foreground since the background and foreground environments are independent of each other.

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

```
:PROG:SOURCE/PLM/INTERRUPT/TEST.P28
```

You can change your default prefix with the command

```
ATTACHFILE :PROG:SOURCE/PLM/INTERRUPT
```

Then, you can refer to the file as simply

```
TEST.P28
```

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

This is the same as entering

:HOME: is a logical name that refers to the same directory as your original default prefix. Therefore, you can change your default prefix as much as you like with ATTACHFILE and return to the original setting by making reference to :HOME:. However, you cannot use ATTACHFILE to change the meaning of :HOME:. Also, you cannot use ATTACHFILE to change the meaning of :CI: and :CO:. Refer to Section 2.6.4.4 for a description of the logical names in a standard system.

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

- A DETACHFILE command (described later in this chapter) dissolves the association between file and logical name.
- The interactive session that specified the ATTACHFILE command terminates processing. This event occurs when a user, in response to the Human Interface prompt, logs off. In this case, the operating system deletes the interactive job.
- A background job created by the standard CLI exits or is KILLED. In this case, only the logical names attached in the background environment are removed.
- A task deletes the connection to the file via a Basic I/O System or Extended I/O System call. In this instance, the logical name remains cataloged in the global directory, but the connection to which it refers does not exist.
- A user forcibly detaches the volume containing the file via the DETACHDEVICE command (described later in this chapter).
- A user removes the volume from the drive. In this instance, the logical name remains cataloged in the global directory, but the connection to which it refers does not exist.

ERROR MESSAGES

- `<logical name>, list of logical names not allowed`

  You entered more than one logical name as input to ATTACHFILE.

- `<pathname>, list of pathnames not allowed`

  You entered more than one pathname as input to ATTACHFILE.

- `<logical name>, logical name not allowed`

  You attempted to attach a file using a logical name :HOME:, :CI:, or :CO:. You cannot change the meaning of these logical names.

- `<logical name>, not a file connection`

  The logical name you specified, <logical name>, is already cataloged in the object directory of the session and does not represent a connection object.

- `<pathname>, not allowed as default prefix`

  You attempted to attach a physical or stream file as your default prefix (:$:). Only named files are valid.

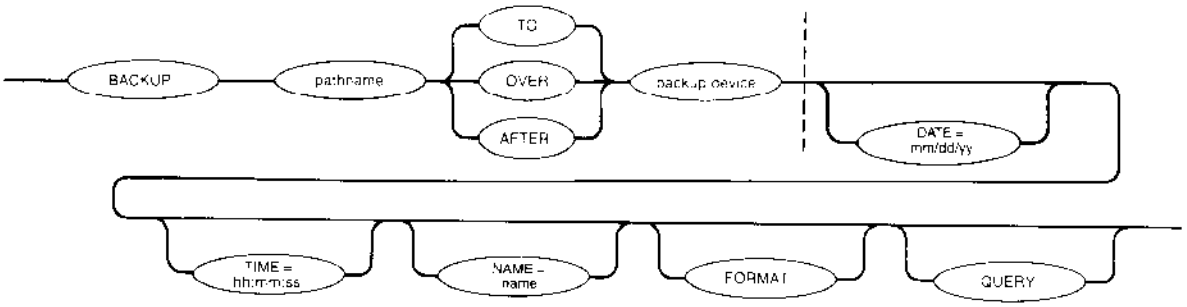
- `<logical name>, too many logical names`

  Your global object directory is full. Therefore ATTACHFILE cannot catalog the logical name in the object directory. Delete some logical names you are no longer using.

This command saves files on a named volume by copying them to a physical volume that serves as a backup storage device. This command provides a way of saving a large volume (a Winchester disk, for example) onto a number of volumes such as diskettes or tape cartridges. Later, you can use the RESTORE command (described later in this chapter) to retrieve these files and copy them to a named volume.

CAUTION

While you use this command, no other activity should be occurring on the volume you are backing up. If other users are accessing the volume during a BACKUP operation, the volume's data could become corrupted, possibly requiring the volume to be reformatted.



x-667-3

INPUT PARAMETERS

pathname — Pathname of a file on the source volume. BACKUP saves all the files starting from this point on the file tree. If you specify the logical name of the device only, BACKUP saves all files in the volume, beginning with the root directory. If you specify a file and not a directory, then only the specified file is saved.

DATE — BACKUP saves all files created or modified on or after the date and time specified with the DATE/TIME parameters. If the DATE parameter is omitted, the date defaults to the current system date. If both date and time parameters are omitted (DATE/TIME), then the date and time default to 1/1/78 and 00:00:00.

mm/dd/yyyy — Form used to specify the DATE.

- mm — Numerical designation for the month. (For example: 1 represents January, 2 represents February, etc.) Must be a digit.
- dd — Numerical designation for the day of the month. Value must be in digits.
- yy — Designation for the year. You enter this as a two- to four-digit number, as follows:

Entered year	Actual year
0 through 77	2000 through 2077
78 through 99	1978 through 1999
100 through 1977	error
1978 through 2099	1978 through 2099
2100 and up	error

TIME — TIME is used in conjunction with the DATE parameter to determine which files to save. If TIME is omitted, the default is 00:00:00. BACKUP saves only those files modified since the specified date or time.

hh:mm:ss — The format for the TIME parameter.

- hh — Hours specified as 0-23
- mm — Minutes specified as 0-59
- ss — Seconds specified as 0-59

NAME — Causes BACKUP to name the set of data you are backing up with this command. Depending on the amount of data being backed up, this named data set can consist of a portion of a single backup volume, or it can span multiple volumes.

If you plan to store multiple data sets on a single backup volume (using the AFTER preposition), it is important to name each data set. Only by naming the data sets can you restore them individually with the RESTORE command.

name — A 1-9 character string that names the volume being backed up during this invocation of the BACKUP command.

To restore a logical volume from a backup volume that contains multiple volumes, you must supply the name of the volume when you invoke the RESTORE command. There is no way to get a listing of these names from the volume itself. Therefore, carefully label the backup volume with the file names present on the volume.

FORMAT — Causes BACKUP to format each volume before writing to it. Interleave is set to one on diskette media. FORMAT should be inserted whenever a new volume is used.

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:

Entry	Action
Y or y	Save the file.
E or e	Exit from BACKUP.
R or r	Continue saving files without further query.
N or n	If a data file, do not save the file; if a directory, do not save the directory or any file in that part of the directory tree. Query for the next file, if any.
Other	Error message and reprompt.

OUTPUT PARAMETERS

TO — Causes BACKUP to send the processed output to a new backup volume. This preposition also causes BACKUP to read the volume label from each newly mounted physical volume in an attempt to determine the volume type. This is an attempt to ensure that the volume is compatible with any previously mounted volumes in the backup set.

OVER — Causes BACKUP to begin writing on each fresh volume without checking the label for compatibility. BACKUP writes over any previous files or directories on the backup volume.

AFTER — Causes BACKUP to search the mounted volume looking for the end of a previous backup operation. BACKUP then appends the file or directory after the previous backup operation. If more volumes are needed to complete the backup operation, then BACKUP behaves as if the TO preposition had been specified for subsequent volumes. If FORMAT was specified, BACKUP formats any new volumes required to finish the backup operation.

:backup device: The 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 tapes or diskettes. For BACKUP to save files from a named volume, you must have read access to the files and to the directories that contain them.

After using the BACKUP command to save named files, you should immediately invoke the RESTORE command with the VERIFY option to make sure that the data on the backup volume has been recorded correctly. Enter the RESTORE command as follows:

```
RESTORE <backup volume> TO :bb: VERIFY
```

where

```
<backup volume>   The logical name of the backup volume you want to
                  verify.
```

RESTORE will only verify that BACKUP produced a restorable backup volume; no files are actually restored.

BACKUP saves the following information for each file:

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

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

```
iRMX II Backup Utility, Vx.y
Copyright <year> Intel Corporation
```

where Vx.y is the version number of the utility.

Once the command line has been scanned, the following message is displayed to indicate what DATE and TIME have been used to save files:

```
All Files Modified After <date> , <time>  Will Be Saved
```

where <date> and <time> are the values you specified in the date and time parameters (or defaults). BACKUP then prompts you to mount the backup volume.

When you use the BACKUP command, you do not have to format a volume previous to issuing the command. BACKUP has a FORMAT parameter which you can use to format any volume while a backup operation is occurring.

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

```
<device>, Mount Backup Volume (name) #<nn>, Enter Y to Continue:
```

where <device> is the logical name of the backup device, (name) is the name of the physical volume set, and <nn> is the identifying number of the requested volume. In response to this message, you place a volume in the backup device and enter one of the following responses:

Entry	Action
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.

A response of Y, y, R, or r causes the BACKUP command to display this buffer summary message:

```
I/O Buffer Summary
    Buffer Size <number>
    Number of Buffers <number>
```

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

If BACKUP detects that a volume cannot be read, that a volume is a named volume, that the volume is a backup volume, or that the volume is a physical volume containing data, the command informs you with one the following messages:

```
<device>, Unrecognized Volume
<device>, Volume Not Correctly Formatted
<device>, Backup Volume <name>, #<nn>, <date>, <time>, Mounted
<device>, Named Volume, <name>, Mounted
```

where <device> is the logical name of the backup device, <name> is the volume name as recorded in the label, <nn> is the volume number of the backup volume, and <date> and <time> are the date and time when the last backup operation was performed. If the situation is appropriate, the command may prompt you with a request to FORMAT or to OVERWRITE the mounted volume in the following way:

```
<device>, Enter Y to Overwrite/Format:
```

In response to this prompt, enter one of the following:

Entry	Action
Y or y	Use the volume as a backup volume.
R or r	Use the volume and do not query for permission again. This is equivalent to specifying 'OVER' on the command line for the rest of the BACKUP operation.
E or e	Exit from the BACKUP command.
N or n	Reprompt for another volume.
Other	Invalid Response; reprompt for entry.

When BACKUP fills a backup volume, it prints the following message:

```
Physical Volume (<name>), #<nn>, Complete
```

BACKUP prompts for additional volumes if it needs them.

After BACKUP finishes, it displays the number of data files and directories it saved, as follows:

```
Summary For Logical Volume (<name>)
   <nn> Data File[s] Saved
   <nn> Director[y] [ies] Saved

   Backup Complete
```

ERROR MESSAGES

Some error messages require a response. If you encounter one of these error messages, enter one of the following:

Entry	Action
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.

The error messages for the BACKUP command are:

- `<backup device>, Backup NOT complete`

  You specified an "E" to exit BACKUP. This message reminds you that the backup operation is not complete. The last file on the last backup volume may be incomplete.

- `<backup device>, Backup Volume (<name>), #<nn>, <date>, <time>, Mounted <backup device>, Enter Y to Overwrite:`

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

- `<backup device>, Cannot Attach Volume`
  `<backup device>, <exception value> : <exception mnemonic>`
  `<backup device>, Mount Backup Volume #<nn>, Enter Y to Continue:`

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

- `<pathname>, <exception value> : <exception mnemonic>, Cannot Back up File`

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

- `<backup device>, Device in Use`
  `<backup device>, <exception value> : <exception mnemonic>`

  The device you specified for the backup device is being used by another process. Continuing would result in damage to the files on the input volume.

- `<backup device>, Error Writing Volume Label`
  `<backup device>, <exception value> : <exception mnemonic>`

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

- `<backup device>, Input and Output are on Same Device`

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

- `<backup device>, Invalid Backup Device`

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

- `<exception value> : <exception mnemonic>, Invalid Date OR Time`

  For either the DATE or TIME parameter, you entered a value that is out of range (such as 31/02/86 or 26:03:62). The message lists the exception code encountered as a result of this entry.

- `Invalid Output Specification`

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

- `<backup device>, <exception value> : <exception mnemonic>`
  `backup device>, Mount Backup Volume #<nn>, Enter Y to Continue:`

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

- `<backup device>, Named Volume <volume name>, Enter Y to Overwrite:`

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

- `<backup device>, Not Correctly Formatted, Enter Y to Format:`

  The backup volume was not correctly formatted.

- `Requested Date/Time Later Than System Date/Time`

  Either the date and time you specified in the BACKUP command are in error or you did not set the system date and time.

- `<pathname>, too many input pathnames`

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

- `<pathname>, too many output pathnames`

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

- `<pathname>, Unable to Complete Directory`

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

- `<backup device>, Unrecognized Volume, Enter Y to Overwrite:`

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

- `<backup device>, Volume Not Formatted`
  `<backup device>, Mount Backup Volume #<nn>, Enter Y to Continue:`

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

- `<backup device>, Write Error On Backup Volume`
  `<backup device>, <exception value> : <exception mnemonic>`

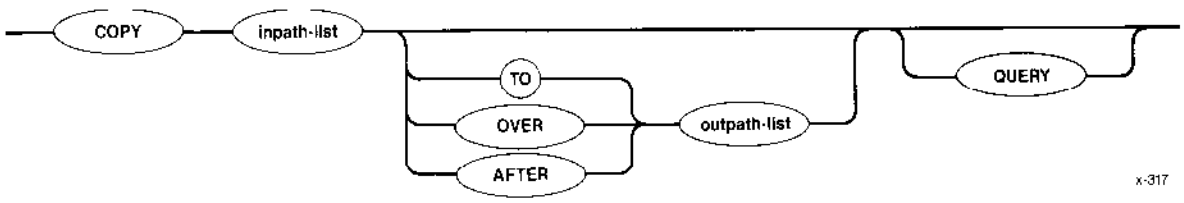
  BACKUP encountered an error condition when writing information to the backup volume. The second line of the message lists the exception code encountered. This error is probably the result of a faulty area on the volume.

- `NAME Required If AFTER Is Selected`

  You must use the NAME parameter when using the AFTER preposition.

This command reads data from the specified input file list and writes the data to the specified destination file or file list.

The format of the command is as follows:



INPUT PARAMETERS

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

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

```
<pathname>, copy TO <out-pathname>?
<pathname>, copy OVER <out-pathname>?
<pathname>, copy AFTER <out-pathname>?
```

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

Entry	Action
Y or y	Copy the file.
E or e	Exit from the 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 inpath list.

OUTPUT PARAMETERS

TO

Writes the listed input files to output files. The specified output file or files should not already exist. If they do, COPY displays the following message:

```
<pathname>, already exists, OVERWRITE?
```

Enter Y, y, R, or r if you wish to write over the existing file. Enter any other character or a carriage return alone if you do not wish to overwrite the existing file. In the latter case, the COPY command will pass over the corresponding input file without copying it, and will attempt to copy the next input file to its corresponding output file.

If you specify multiple input files and a single output file, COPY appends the remaining input files to the end of the output file.

OVER

Writes the input files over (replaces) the existing output files on a one-for-one basis, regardless of file size. If an output file does not already exist, its corresponding input file is written to a new file with the corresponding output file name. If you specify multiple input files and a single output file, COPY appends the remaining input files to the end of the output file.

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 you omit the preposition and outpath-list parameters, COPY displays the output at your console screen (TO :CO:).

DESCRIPTION

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

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

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

```
<pathname> copied TO <out-pathname>
```

```
<pathname> copied OVER <out-pathname>
```

```
<pathname> copied AFTER <out-pathname>
```

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

```
<pathname> copied AFTER <output-file>
```

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

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

You cannot successfully use COPY to copy a directory to a data file or to another directory. Although a dirctory 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

This would copy data file A to a different volume, directory, and filename.

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

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

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

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

The COPY command cannot be used with tape cartridges.

ERROR MESSAGES

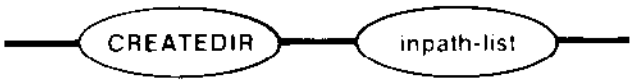
- `<pathname>, output file same as input file`

  You attempted to copy a file to itself.

- `<pathname>, UPDATE or add access required`

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

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



INPUT PARAMETER

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

DESCRIPTION

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

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

```
<directory-name>, directory created
```

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

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

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

ERROR MESSAGES

- `<directory-name>, file already exists`

  The pathname of the directory to be created already exists.

- `<file-name>, file does not exist`

  One of the directories in the specified pathname does not exist.

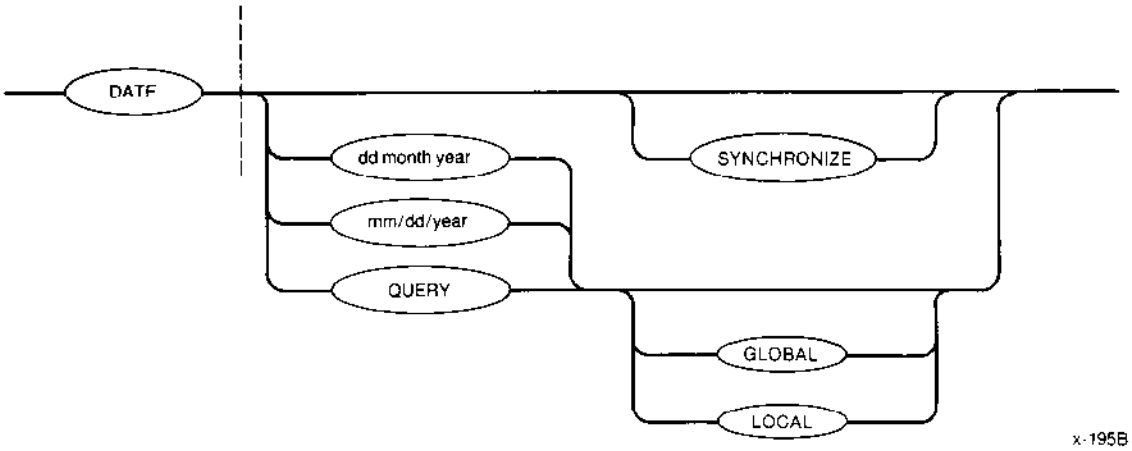
- `<directory-name>, invalid file type`

  One of the directories specified in the pathname is not a valid directory.

- `<directory-name>, logical name does not exist`

  The logical name you entered as the directory name does not exist.

This command sets a new system date or displays the current date. This command sets the date portion of either the local or global time-of-day clock. The format is as follows:



INPUT PARAMETERS

dd — Two-digit number that specifies the day of the month. Both digits are not required to set this parameter.

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

mm — Numerical designation for the month (for example: 1 represents January, 2 represents February, etc.). You can use this form for specifying the month only when using the "mm/dd/year" format. Both digits are not required to set this parameter.

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

Entered year	Actual year
0 through 77	2000 through 2077
78 through 99	1978 through 1999
100 through 1977	error
1978 through 2099	1978 through 2099
2100 and up	error

QUERY — Causes DATE to display the current date, time and clock type followed by the prompt

```
DATE:
```

DATE continues to issue this prompt until you enter a valid date or the letter E (or e) to exit.

GLOBAL — Applies only to systems with hardware clock/calendar components. Such clock/calendar components are usually powered by batteries so they continue keeping time when power to the system is turned off. These clock/calendar components are referred to as global clocks. This parameter causes DATE to display or set the date portion of the global time-of-day clock. Any user can display the current value of the global clock, but only the system manager can set the global clock. If the global clock is modified, the local clock automatically takes on the new value of the global clock. LOCAL is the default if the LOCAL and GLOBAL parameters are omitted.

DATE displays an error message if you specify this parameter and your system does not have a global clock/calendar.

LOCAL — Causes DATE to display or set the date portion of the local time-of-day clock maintained by the operating system. All users can display the current value of the local clock or set it to a new value. LOCAL is the default if the LOCAL and GLOBAL parameters are omitted.

SYNCHRONIZE — Applies only to systems with global clock/calendars. This parameter causes DATE to set the date portion of the local time-of-day clock to the current date value in the global clock. If you are modifying the global clock, this parameter is unnecessary.

DATE displays an error message if you specify this parameter and your system does not have a global clock/calendar.

DESCRIPTION

Entering this command causes the current date and time, and the clock type to be displayed. If you set only one or two date parameters, the omitted parameters are replaced by their defaults. If you enter only one parameter, it is assumed to be the day. Two parameters represent day and month. For example, assume the current date in the system is 9 Sept 86. If you enter

DATE will display

```
18 Sep 86, <current time>
```

You must separate the day, month, and year entries with single blanks.

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

```
dd month yy, hh:mm:ss clocktype (global or local)
```

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

If you have a system without a global clock/calendar (such as a System 310), whenever you start up or reset the operating system, the date is automatically set to the date you last accessed the :SYSTEM: directory. You can reset the date to any acceptable value.

If your system has a global clock/calendar and the operating system is configured to recognize it, the local clock is automatically set to the date maintained in the global clock whenever you turn on or reset your system.

The DATE command enables you to set and/or display the date portion of two time-of-day clocks: the local clock and the global system clock. You access the local clock by specifying the LOCAL parameter; you access the global clock by specifying the GLOBAL parameter. If neither LOCAL nor GLOBAL is specified, the local clock is accessed by default. Any user can display the date (and time) portion of the local and global clocks. However, only the system manager can set the date portion of the global clock. If the system manager sets the global clock to a new value, the local clock will automatically be set to that value.

ERROR MESSAGES

- `<date>, invalid date`

  You entered an invalid date. This error could result from specifying a day that is invalid for the month you specified (such as 31 FEB 86), entering characters for the year parameter that do not fall into the legitimate ranges listed under the year parameter, entering a month parameter that does not uniquely identify the month, or entering invalid characters.

- `<parameter>, invalid syntax`

  You specified an illegal combination of parameters. For example, you may have entered a date with the QUERY option.

- `only the system manager may set the global clock`

  You specified the GLOBAL parameter, but you are not the system manager.

- `<exception value> : <exception mnemonic>, getting system time`

  You specified the GLOBAL or SYNCHRONIZE parameter, but there is no global clock in the system.

- `E$SHARE, global clock busy`

  You attempted to access the global time-of-day clock while another job was accessing it. Try the command again.

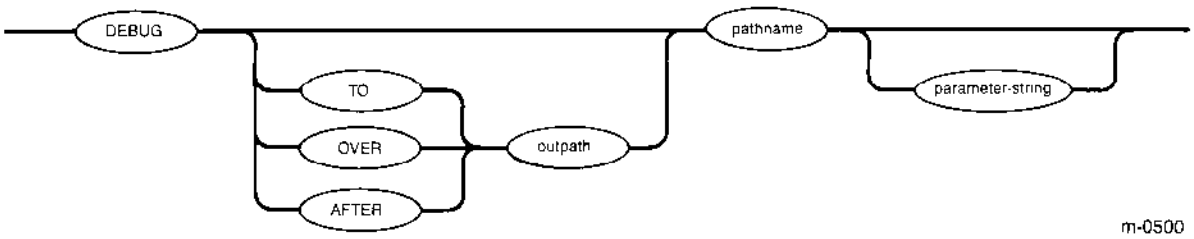
- `E$INVALID$DATE, global date read was invalid`

  The date returned from the global clock was invalid. This condition will usually occur when the global clock has never been initialized or when power to the clock has been interrupted. The BIOS system call GET$GLOBAL$TIME sets the date to a valid date, 1 Jan 1978, which the DATE command then displays.

- `E$INVALID$TIME, global time read was invalid`

  The time returned from the global clock was invalid. This condition will usually occur when the global clock has never been initialized or when power to the clock has been interrupted. The BIOS system call GET$GLOBAL$TIME sets the time to a valid time, which the DATE command then displays.

This command allows you to debug your iRMX II application jobs if your system is configured with the iSDM monitor.



INPUT PARAMETERS

pathname	Pathname of the file containing the application program to be debugged.
parameter-string	String of required and optional parameters used by the application program to be debugged.

OUTPUT PARAMETERS

TO	Writes the debug information to the named new output file. The specified output file should not already exist. If it does, DEBUG displays the following message: `<pathname>, already exists, OVERWRITE?` Enter Y, y, R, or r if you wish to write over the existing file. Enter any other character or a carriage return alone if you do not wish to overwrite the existing file.
OVER	Writes the debug information over (replaces) the existing output file, regardless of file size. If an output file does not already exist, the debug information is written to a new file with the corresponding output file name.
AFTER	Appends the debug information to the current data in the existing output file. If the output file does not already exist, all debug information will be concatenated into a new file with the listed output file name.

outpath-list — A pathname for the output file or logical device to receive the debug information. The most commonly used names are :lp: and :fn:file where :fn: is the logical name of a secondary storage device onto which the debug information will be written. If no output file is specified, DEBUG displays the output at the console screen (TO :CO:).

DESCRIPTION

DEBUG loads your specified application program into main memory, puts the debug information into an output file (or on the screen, if you did not specify an output filename), and transfers control to the system monitor. You can then use the monitor to single-step, display registers, and set breakpoints within the program. Refer to the *iSDM System Debug Monitor Reference Manual* for more information.

The DEBUG command cannot be used to debug CLI level commands, only HI commands and application programs.

When you invoke the DEBUG command with no output file, it displays the following message:

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

where <pathname> is the pathname of the file containing the application job to debug. Then DEBUG loads the application job and displays the debug information on the console screen (or writes it to an output file if you specified one). Figure 4-2 shows an example of this output. After writing the debug information, DEBUG waits until you indicate that you're ready to enter the monitor by entering <RETURN>. This allows you to access the optional debug file from a remote system (using iRMX-NET) to aid in the debug process.

As Figure 4-2 shows, the first line of the debug information lists the token for the job that was created. The remaining lines list the selector portions of all segments (under the heading BASE) assigned by BND286 when the code was bound. The LDT(n) values are the same as those that appear on the bind map. Therefore, you can match the selector values shown in this display with the offset values shown in the bind map to determine the exact location of a symbol listed in the bind map. Refer to the *iAPX 286 Utilities User's Guide* for information about BND286 and the bind map.

```
        SEGMENT MAP FOR JOB:  2250

  NAME    BASE    NAME    BASE    NAME    BASE    NAME    BASE

  LDT(2)  2E40    LDT(3)  2E30    LDT(4)  2C08    LDT(5)  2CE0
  LDT(7)  2220    LDT(8)  2158

  Interrupt 3 at xxxx:yyyy
  ..
```

Figure 4-2. Sample DEBUG Display

If you invoke the DEBUG command with output redirection, the information displayed in Figure 4-2 is written to the output file specified. After the debug information has been written to the output file, the following message is displayed:

```
Please hit <RETURN> when you are ready to break to the monitor:
```

The system breaks to the monitor immediately after a <RETURN> is entered.

When DEBUG executes, the monitor in your system disables interrupts. This causes the time-keeping function to stop when code is not executing. This slowing of the timing function

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

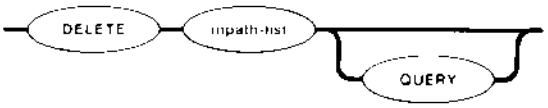
When DEBUG is invoked to debug an application program, it loads the application program into its own dynamic memory. As a result of this process, the application program obtains dynamic memory from the memory pool of DEBUG, not from the memory pool of the user session. Because DEBUG uses a different set of default values than the CLI, it is possible that the program may behave differently than when it is run independently.

ERROR MESSAGE

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

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

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



x-319

INPUT PARAMETERS

inpath-list — One or more pathnames for the named data files or empty directories to be deleted. Multiple pathname entries must be separated by commas. Separating blanks are optional.

QUERY — Causes the DELETE command to ask for 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:

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

DESCRIPTION

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

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

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

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

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

```
<pathname>, deleted
```

WARNING

Be careful when using wildcards with the DELETE command. For example, entering DELETE *,a instead of DELETE *.a can erase all your files. As a precaution, the DELETE command displays this message before deleting any files in this case:

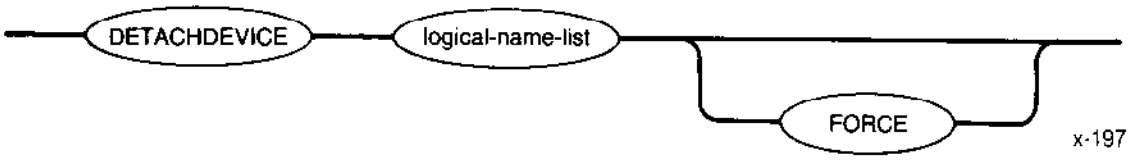
Do you really want to delete your files? (y or [n])

ERROR MESSAGE

- `<pathname>, DELETE access required`

  You do not have permission to delete the specified file.

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



INPUT PARAMETER

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

DESCRIPTION

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

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

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

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

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

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

```
<logical-name>, detached
```

NOTE

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

ERROR MESSAGES

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

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

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

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

- `<logical name>, device has outstanding file connections`

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

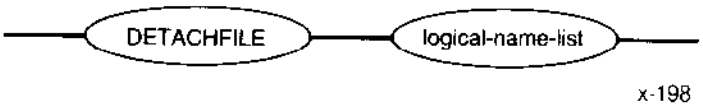
- `<logical name>, device is in use`

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

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

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

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



PARAMETERS

logical-name-list — List of logical names, separated by commas, that represent the files to be detached. Each logical name must contain 1 to 10 characters or 1 to 12 characters if surrounding colons are used. Colons surrounding each logical name are optional; however, if you use colons, you must use matching colons.

DESCRIPTION

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

```
<logical name>, detached
```

where <logical name> is the name you specified.

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

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

ERROR MESSAGES

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

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

- `<logical name>, logical name does not exist`

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

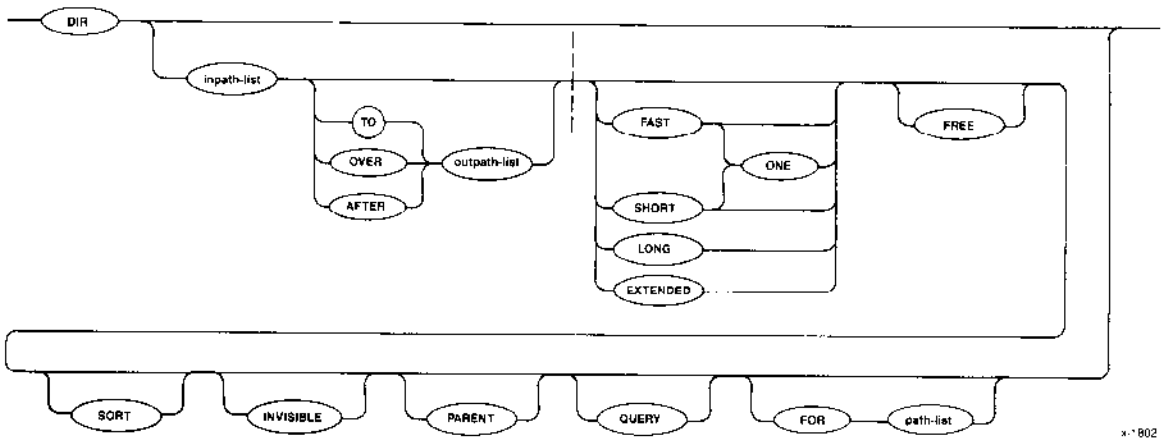
- `<logical name>, logical name not allowed`

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

- `<logical name>, not a file connection`

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

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



INPUT PARAMETERS

inpath-list	One or more pathnames of the directories to be listed (the pathnames can represent data files if the PARENT parameter is also specified). Multiple directory pathname entries must be separated by commas. Separating blanks are optional. If no pathname is specified, the user's default directory is listed.
FAST	Lists only the filenames and directory names in the directory. The output format contains five columns of filenames unless you also specify the ONE parameter (see Figure 4-3 at the end of this command description). FAST is assumed if you omit the listing format.
SHORT	Lists the file information in a two-column format unless you also specify the ONE parameter (see Figure 4-4 at the end of this command description).
ONE	Lists the output of a FAST or SHORT listing in single-column format. ONE is the assumed number of columns for EXTENDED or LONG listings.

LONG	Lists file information in a one-line format (see Figure 4-5 at the end of this command description).
EXTENDED	Lists all available information for each data file or directory file in the directory. The first line for each file is the same as for the LONG form. The next line contains the last access date, the creation date, the last modified date, and the accessor list. The listing is in a double-column format (see Figure 4-6 at the end of this command description).
FREE	Lists the amount of free space available on the volume containing the given directory. The listing shows the number of free files, free volume blocks, and free bytes.
SORT	Lists the filenames and directories in the directory in alphabetical order.
INVISIBLE	Lists the invisible files (those beginning with the characters "R?" or "r?") in addition to the rest of the files in the directory. If you omit this parameter, DIR does not display invisible files.
PARENT	Causes DIR to display an entry for the directory specified in the inpath-list in addition to the files contained in the directory. This parameter is useful for obtaining information about the root directory of a volume when using the LONG or EXTENDED parameters.
QUERY	Causes the DIR command to prompt you for permission to list a directory by issuing the following message:

```
<pathname>, DIR?
```

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

Entry	Action
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.

FOR path-list	Selects only those files that match one of the names in the path-list. The path-list can include wildcard file designators.

OUTPUT PARAMETERS

TO	Copies the directory listing to the specified destination data file. If the destination file already exists, DIR displays the following information: `<pathname>, already exists, OVERWRITE?` Enter Y, y, R, or r if you wish to delete the existing file. Enter any other character if you do not wish to delete the file. If you omit the TO/OVER/AFTER preposition and the output pathname, TO :CO: is assumed.
OVER	Copies the directory listing to the specified output file and writes over (replaces) its previous contents.
AFTER	Appends the directory listing to the current contents of the specified output file.
outpath-list	One or more pathnames of the files to receive the directory listing. Multiple pathname entries must be separated by commas. Separating blanks are optional. If you omit the preposition and the outpath-list, the default destination is the user's console screen (TO :CO:).

DESCRIPTION

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

The amount of information listed for each file depends upon what listing format you specify (FAST, SHORT, LONG, or EXTENDED). The end of the SHORT, LONG, and EXTENDED DIR listings show the amount of space used (first line) by the files and the amount of free space left over (second line).

An example of each type of listing format is provided at the end of the DIR command description in Figures 4-3 through 4-6 respectively. Table 4-6, which follows the figures, provides an explanation of the directory listing headings.

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

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

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

```
-DIR alpha <CR>

03 MAR 86  04:25:40
DIRECTORY OF alpha  ON VOLUME myvol
fname1  fname2  fname3  fname4  fname5
fname6  fname7  fname8  fname9  fname10
fname11 . . .
.
.
.
```

Figure 4-3. FAST Directory Listing Example (Default Listing Format)

```
-DIR mydirectory2 S <CR>

03 MAR 86   21:55:24
DIRECTORY OF mydirectory2 ON VOLUME myvol
  NAME          AT  ACC  BLKS   LENGTH      NAME          AT  ACC  BLKS  LENGTH
append              -R--  2     1,425       alpha.obj         DRAU  3     2,871
REFERENCE     DR    -L--  1     10          DATA          DR  DLAC  1     4
LEMONADEIT          DRAU  978   1,000,912
time                DRAU  6     5,374       detachdevice      DRAU  4     3,414
test                -R--  5     4,415       schedule          ---U  7     6,976
testprog.a28        -RA-  2     2,040       DATABASE.LST      -RAU  11    10,336
EXPERIMENTAL  DR    -LAC  1     20          BACKUP        DR  DLAC  1     10

     13 FILES      995 BLKS       1,014,196 BYTES

     33 FILES    3,000 BLKS       3,072,000 BYTES FREE
```

Figure 4-4. SHORT Directory Listing Example

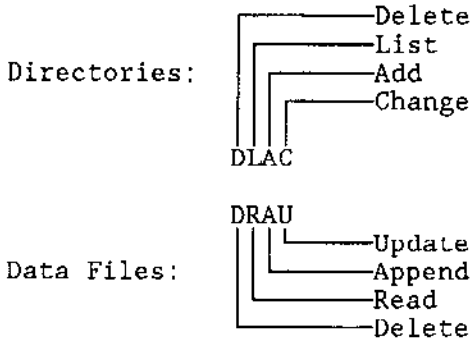
```
-DIR mydirectory1 L <CR>

03 MAR 86   21:55:24
DIRECTORY OF mydirectory1 ON VOLUME myvol
                                                    GRAN
   NAME       AT      ACC   BLKS    LENGTH    VOL FIL   OWNER      LAST MOD
ed                    -R--     11     1,057  1,024   1   # 47      02 MAR 86
programs     DR       DL--     30    30,185  1,024   1   # 47      03 MAR 86
fmat                  DRAU      1        39  1,024   1   # WORLD   08 NOV 85
OBJFILE               ---U      3     2,895  1,024   1   # 47      18 DEC 85
ALPHA1.P28            DLAC      2     1,304  1,024   1   # 50      22 OCT 85
ALPHA1.MP1            DLAC      6     5,397  1,024   1   # 50      22 OCT 85
manuals      DR       -L--      1       304  1,024   1   # 47      02 JUL 85

      7 FILES      54 BLKS          41,181 BYTES

     33 FILES  3,000 BLKS       3,072,000 BYTES FREE
```

Figure 4-5. LONG Directory Listing Example

```
-DIR mydir E <CR>

03 MAR 86   21:55:24
DIRECTORY OF mydir ON VOLUME myvol
                                                    GRAN
   NAME    AT   ACC   BLKS      LENGTH    VOL  FIL OWNER          LAST MOD
programs   DR   DL--    30      30,185   1,024   1  # 47          03 MAR 82
                  CREATION: 01 JAN 86  04:05:44     ACCESSORS        ACC
                  LAST ACC: 03 MAR 86  05:52:33     # 47             DL--
                  LAST MOD: 03 MAR 86  05:52:33     # 50             -LA-
                                                    # 82             -L--
ed                -R--    11       1,057    1,024   1  # 47          02 MAR 82
                  CREATION: 11 NOV 85  12:24:05     ACCESSORS        ACC
                  LAST ACC: 02 MAR 86  14:22:16     # 47             -R--
                  LAST MOD: 02 MAR 86  14:22:16
fmat              DRAU     1          39    1,024   1 WORLD          08 NOV 81
                  CREATION: 01 NOV 85  08:54:39     ACCESSORS        ACC
                  LAST ACC: 03 MAR 86  14:56:59     WORLD            DRAU
                  LAST MOD: 08 NOV 85  20:44:01
testdir    DR   DLAC     1          32    1,024   1  # 47          01 MAR 82
                  CREATION: 02 FEB 85  15:02:42     ACCESSORS        ACC
                  LAST ACC: 03 MAR 85  09:32:53     # 47             DLAC
                  LAST MOD: 01 MAR 85  13:13:07     # 50             -LA-
                                                    WORLD            -L--

          4 FILES     43 BLKS          32,213 BYTES

          33 FILES  3,000 BLKS      3,072,000 BYTES FREE
```

Figure 4-6. EXTENDED Directory Listing Example

Table 4-6. Directory Listing Headings

Heading	Meaning
NAME	Up to 14-character file name.
AT	File attribute, where: DR = Directory MP = Bit map file blank = Data file
ACC	File access rights of the user who entered the DIR command, where: 
BLKS	Nine-digit number (five digits on SHORT listing) giving the volume-granularity units allocated to the file. On the SHORT display, if the number of digits exceeds five, DIR displays the file in the nine-digit form (see the LEMONADEIT file in Figure 4-4).
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 LEMONADEIT file in Figure 4-4).
VOL	Five-digit number giving the volume granularity in bytes.
FIL	Three-digit number giving the granularity of the file in multiples of volume granularity.
OWNER	14-character, alphanumeric owner name.
LAST MOD	Date of last file modification.
LAST ACC	Date of last file access.
CREATION	Date of file creation.
ACCESSORS	User IDs of users who have access to the file.
ACC	Access rights of the corresponding user. The format of this field is identical to ACC as described previously.

ERROR MESSAGES

- `no directory files found`

  None of the files you specified were directories.

- `<pathname>, READ access required`

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

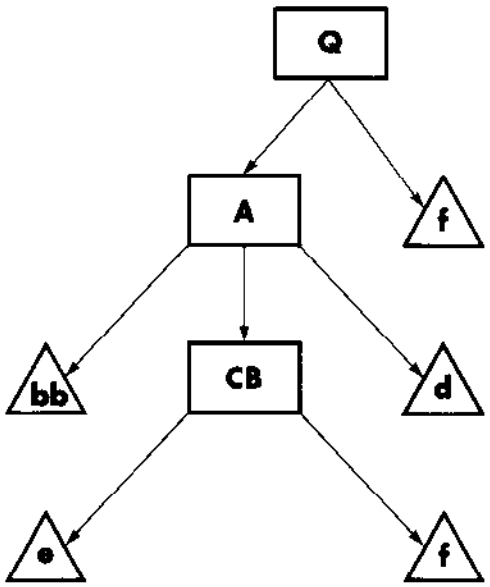
- `<pathname>, UPDATE or ADD access required`

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

EXAMPLES

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

Assume you have the following directory structure for your files:



x-324

Example 1:

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

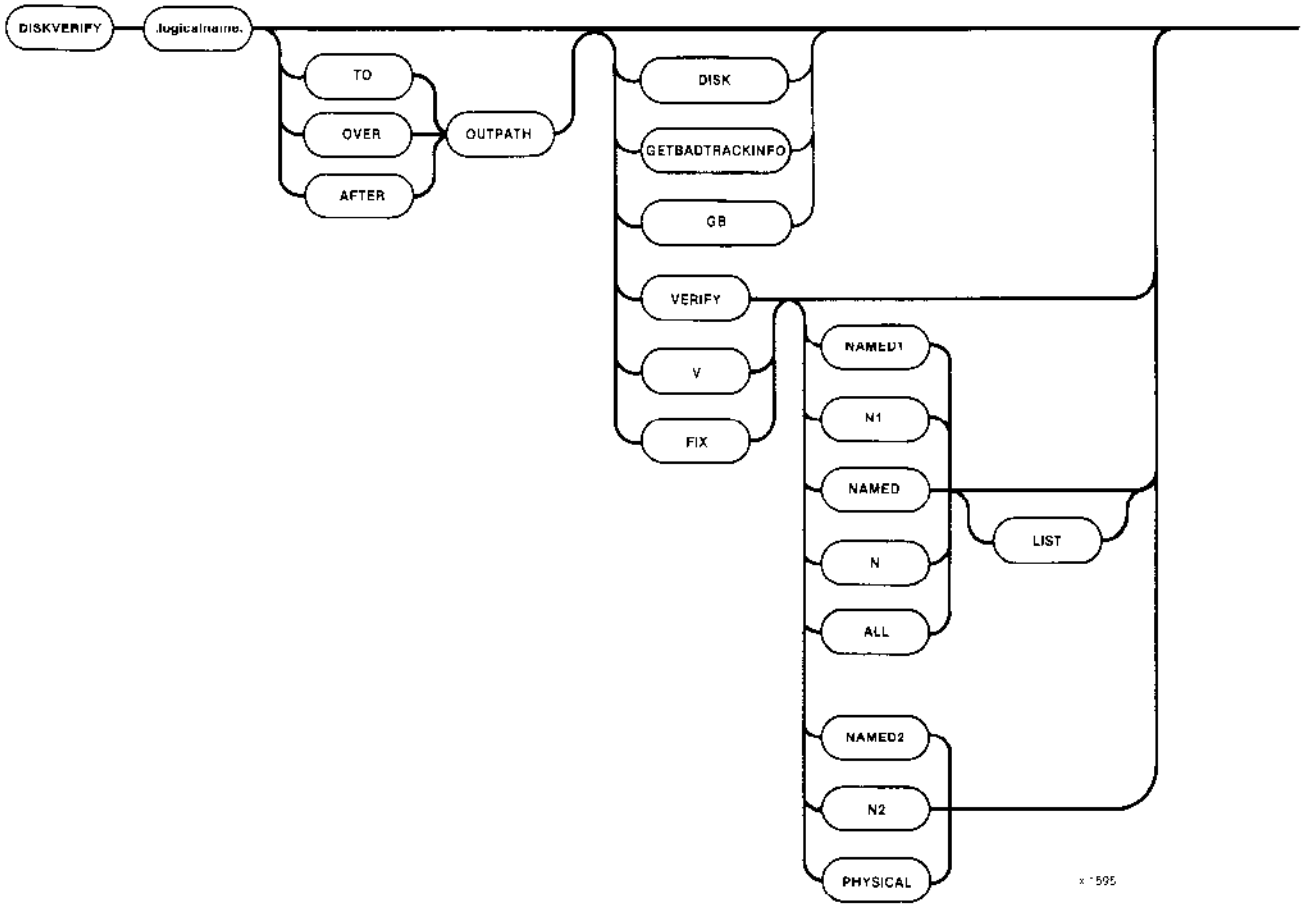
Pathname	Files Listed
omitted	A, f
f	not allowed because f is a data file
A	bb, CB, d
A/d	not allowed because d is a data file
A/CB	e, f
A/CB/e	not allowed because e is a data file

Example 2:

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

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

This command invokes the disk verification utility which verifies the data structures of iRMX II 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 to be verified.

If you don't specify any parameters, 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 *Extended iRMX II Disk Verification Utility Reference Manual.*

DISK	Displays the attributes of the volume (such as type of volume, device granularity, block size, number of blocks, interleave factor, extension size, volume size, and number of fnodes) and returns control to you at the Human Interface level. You can then enter any Human Interface command, provided that the device verified is not the system disk. You must reboot your system if the device verified is the system device.
GETBADTRACKINFO or GB	Reads the bad track information from the volume and displays it. Bad track information that is redirected to a file can be used as input to the FORMAT command by removing the header information.
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, provided the device verified is not the system device. You must reboot your system if the device verified is the system device.

FIX

Performs the same functions as VERIFY. In addition, FIX tries to fix several types of inconsistencies on the volume after performing the verification. You should be careful when using FIX as it changes the data on the disk (which may prove dangerous). For example, during NAMED1 verification, FIX corrects the checksums on fnodes with bad checksums. However, an fnode with a badchecksum may indicate another problem with the fnode which should not be ignored. As a result, it is recommended that you use FIX only after performing the following steps.

1. Use DISKVERIFY with the VERIFY option.
2. Examine the output and the problems on the volume to determine the type of "fix" needed.
3. If the problems can be fixed using DISKVERIFY, run DISKVERIFY with the FIX option to correct the problem.

NAMED1 or N1

VERIFY or FIX option that applies to named volumes only. This option checks the fnodes of the volume to ensure that they match the directories in terms of file type and file hierarchy. (Refer to the description of the FORMAT command for more information about fnodes.) This option also checks the information in each fnode to ensure that it is consistent. As a result of this option, DISKVERIFY displays a list of all files on the volume that are in error, with information about each file.

When used with FIX, the NAMED1 option corrects bad checksums and attaches orphan fnodes to their parents. Refer to the *Extended iRMX II Disk Verification Utility Reference Manual* for more information.

NAMED2 or N2	VERIFY or FIX 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. When used with FIX, the NAMED2 option saves the correct bit maps that were constructed during N2 verification, on the volume. It also removes fnodes with multiple references from their illegal parents. Refer to the *Extended iRMX II Disk Verification Utility Reference Manual* for more information.
NAMED or N	VERIFY or FIX option that performs both the NAMED1 and NAMED2 verification functions on a named volume. If you omit the VERIFY option, NAMED is assumed.
ALL	VERIFY or FIX option that applies to both named and physical volumes. For named volumes, this option performs both the NAMED and PHYSICAL verification functions. For physical volumes, this option performs only the PHYSICAL verification function.
PHYSICAL	VERIFY or FIX option that applies to both named and physical volumes. This option reads all blocks on the volume and checks for I/O errors.
LIST	A control that you can use with any option that activates NAMED1 verification (NAMED, NAMED1, or ALL). When you use this option, the file information generated by VERIFY or FIX is displayed for every file on the volume, even if the file contains no errors. Refer to the *Extended iRMX II Disk Verification Utility Reference Manual* for more information.

OUTPUT PARAMETERS

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

```
<pathname>, already exists, OVERWRITE? ·
```

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

If no preposition is specified, TO :CO: is assumed.

OVER — Copies the output from the disk verify utility over the specified file.

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

outpath — Pathname of the file to receive the output from DISKVERIFY. 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 CONNECTION error message.

DESCRIPTION

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

```
iRMX II Disk Verify Utility, Vx.y
Copyright <year> Intel Corporation
```

where Vx.y is the version number of the utility.

If you specify the VERIFY, FIX, GETBADTRACKINFO, or DISK parameter in the DISKVERIFY command, the utility performs the operation specified in the parameter and copies the output to the console (or to the file specified by the outpath parameter). Refer to the *Extended iRMX II Disk Verification Utility Reference Manual* for a description of the output. After generating the output, the utility returns control to the Human Interface, which prompts you for more Human Interface commands. The following is an example of a DISKVERIFY command that uses the VERIFY option:

```
-DISKVERIFY :F1: VERIFY NAMED2 <CR>
iRMX II Disk Verify Utility,  Vx.y
Copyright <year> Intel Corporation

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

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

The following is an example of a DISKVERIFY command with the FIX option. It performs both named and physical verification of a named volume and corrects the problems on the volume.

```
-DISKVERIFY :F1: FIX ALL <CR>

iRMX II Disk Verify Utility, Vx.y
Copyright <year> Intel Corporation

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

'NAMED1' VERIFICATION
'NAMED2' VERIFICATION
   BIT MAPS O.K.
'PHYSICAL' VERIFICATION
   NO ERRORS
   free fnode map saved
   free space map saved
save bad block map? <y>
   bad block map saved
-
```

The following example uses the GETBADTRACKINFO option. (This example may be useful when migrating from a 214 controller to a 224A controller.)

```
-DISKVERIFY :SD: TO :F1:WORK/BT GB <CR>
iRMX II Disk Verify Utility, Vx.y
Copyright <year> Intel Corporation

Bad track information

cyl   head   sector
0034  03     00
0043  02     00
0316  00     00
```

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

```
-DISKVERIFY :F2: DISK <CR>
iRMX II Disk Verify Utility, Vx.y
Copyright <year> Intel Corporation

                Device name = WF0
      named disk, volume name = UTILS
          device granularity = 0080
                  block size = 0080
            number of blocks = 0000072D
       number of free blocks = 00000408
                 volume size = 0003E900
                  interleave = 0005
             extension size = 03
           number of fnodes = 0038
      number of free fnodes = 0022
         save area reserved = no
-
```

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

```
-DISKVERIFY :F1: <CR>
iRMX II Disk Verify Utility, Vx.y
Copyright <year> Intel Corporation
*
```

After you receive the asterisk prompt, you can enter any of the DISKVERIFY commands listed in the *Extended iRMX II Disk Verification Utility Reference Manual*.

ERROR MESSAGES

- `argument error`

  The option you specified is not valid.

- `command syntax error`

  You made a syntax error when entering the command.

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

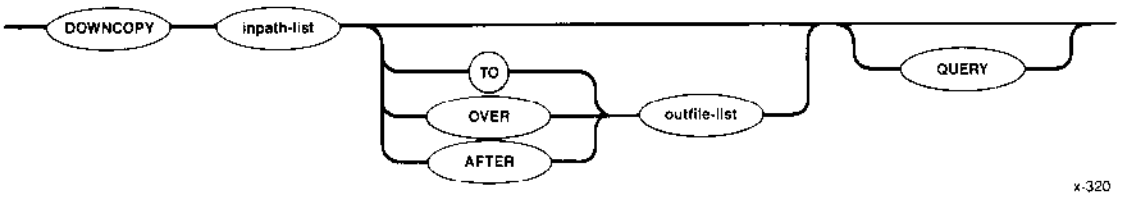
  When the disk verify utility computed the size of the volume based on the physical name used to attach, the size it computed did not match the information recorded in the iRMX II volume label. It is likely that the volume label contains invalid or corrupted information. This error is not a fatal error, 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 verify utility to modify the volume label. Refer to the *Extended iRMX II Disk Verification Utility Reference Manual* for more information about the disk verify utility commands.

- `not a named disk`

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

The NAMED1, NAMED2, and PHYSICAL verification options can also produce error messages. Refer to the *Extended iRMX II Disk Verification Utility Reference Manual* for more information about these messages.

This command copies files from a volume on an iRMX II secondary storage device to a volume on a Series II, II, or IV secondary storage device. The format is as follows:



INPUT PARAMETERS

inpath-list — One or more iRMX II pathnames for files, separated by commas, that are to be copied to development system secondary storage. Separating blanks between pathnames are optional. The files can 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 II file to the listed destination file. Depending on which preposition you specify (TO, OVER, or AFTER), the Human Interface prompts with one of the following queries:

```
<pathname>, copy down TO <outfile>?
<pathname>, copy down OVER <outfile>?
<pathname>, copy down AFTER <outfile>?
```

Enter one of the following in response to the query:

Entry	Action
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 the sequence.

OUTPUT PARAMETERS

TO — Reads iRMX II files and copies them TO new development system files in the listed sequence. If the specified output files already exist in the development system directory when the TO parameter is used, DOWNCOPY displays the following message:

```
<filename>, already exists, OVERWRITE?
```

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

If no preposition is specified, TO :CO: (console screen) is assumed. If more input files than output files are specified, the remaining input files are appended to the end of the last-specified output file.

OVER — Copies the iRMX II input files OVER the existing development system files in the specified sequence. If you specify multiple input files and one output file, DOWNCOPY displays an error message.

AFTER — Copies the iRMX II input files, in sequence, AFTER the end of data on the existing development system destination files.

outfile-list — One or more development system filenames for the output files. Multiple filenames must be separated by commas. Separating blanks are optional. If the preposition and output file parameters are not used in the command line, the default is :CO:.

DESCRIPTION

The DOWNCOPY command cannot be used to copy directories from an iRMX II system to a Series II, II, or IV Microcomputer Development System; only files can be copied.

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

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

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

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

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

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

When the DOWNCOPY command is executing, the monitor disables interrupts. This event affects services such as the time-of-day clock. Also, the operating system is unable to receive any characters that you type-ahead while the monitor is disabling interrupts.

ERROR MESSAGES

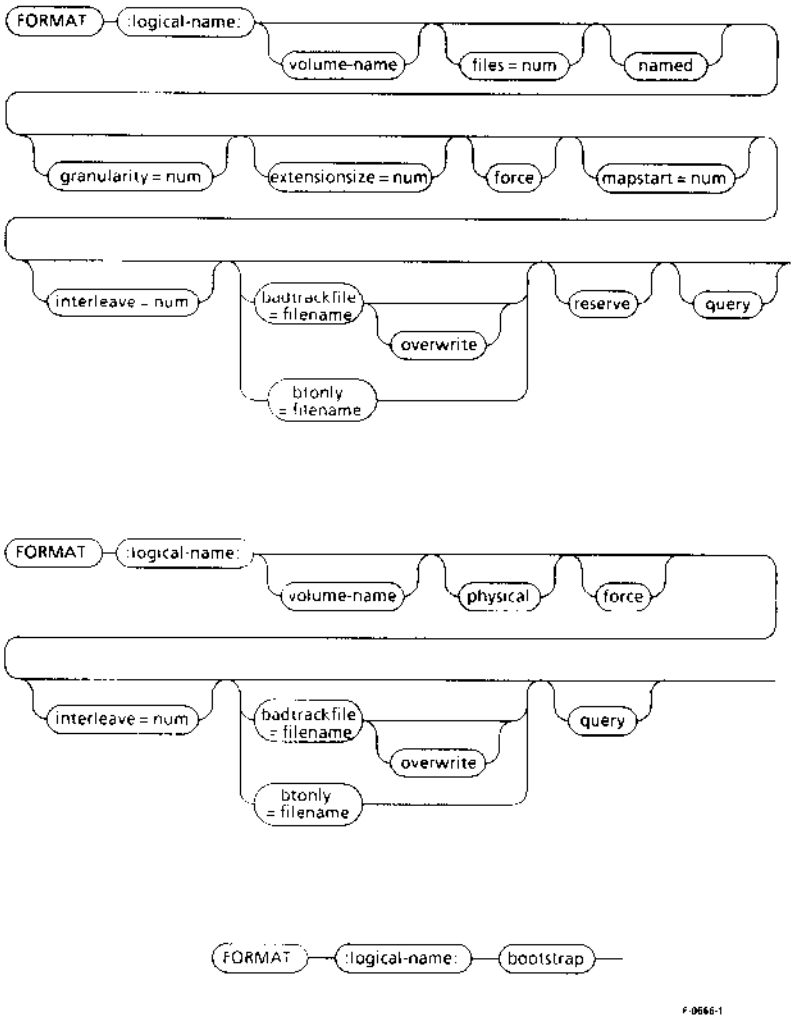
- `<pathname>, ISIS ERROR: <nnn>`

  A development system error occurred when DOWNCOPY tried to transfer the file to the Microcomputer Development System. Refer to the appropriate operating and and programming guide for a description of the resulting error code.

- `ISIS link not present`

  The iRMX II system is not connected to the development system via the monitor.

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



INPUT PARAMETERS

Parameters that can be abbreviated have the abbreviation(s) listed under the complete parameter name.

:logical-name:	Logical name of the physical device-unit to be formatted. You must surround the logical name with colons. Also, you must not leave space between the logical name and the succeeding volume name parameter.

volume-name	An alphanumeric ASCII name, of up to six characters, without embedded blanks, to be assigned to the volume. If you include this parameter, you must not leave spaces between the logical name and the volume name.
FILES = num FI	Defines the maximum decimal number of user files that can be created on a NAMED volume. (This parameter is not meaningful when formatting a PHYSICAL volume and is ignored if specified for such volumes.) FORMAT uses the information specified in this parameter to determine how many structures (called fnodes) to create on the NAMED volume. The range for the FILES parameter is 1 through 65,528, although the maximum number of user files you can define depends on the settings of the GRANULARITY and EXTENSIONSIZE parameters (as explained in the "Description" portion of this command write-up). When you use this parameter, FORMAT creates five additional fnodes for internal system files (six if you include the RESERVE parameter), and an additional fnode for the root directory. If not specified, the default is 200 user files.
FORCE	Forcibly deletes any existing connections to files on the volume before formatting the volume. If you do not specify FORCE, you cannot format the volume if any connections to files on the volume still exist.
MAPSTART = num MS M	Gives the volume block number where the fnodes file, bit map files, and the root directory should start. The size of the block is set by the GRANULARITY parameter. If no number is given, the operating system puts the fnodes file in the center of the volume. If the number is too low, the operating system places the map files at the lowest available space on the volume.

GRANULARITY = num
GU
G

Volume granularity; the minimum number of bytes to be allocated for each increment of file size on a NAMED volume. (This parameter is not meaningful for PHYSICAL volumes, and is ignored if specified for such volumes.) FORMAT rounds the value you specify up to the next multiple of the device granularity. Then it places the decimal number in the header of the volume, where it becomes the default file granularity when a file is created on the volume.

The range is 1 through 65,535 (decimal) bytes, although the maximum allowable volume granularity depends on the settings of the FILES and EXTENSIONSIZE parameters (as explained in the "Description" portion of this write-up).

If not specified, the default granularity is the device granularity. Once the volume granularity is defined, it applies to every file created on that volume.

NOTE

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

EXTENSIONSIZE=num ES E	Size, in bytes, of the extension data portion of each file. (This parameter is not meaningful for PHYSICAL volumes, and is ignored if specified for such volumes.) The range is 0 through 255 (decimal), although the maximum allowable extension size depends on the settings of the FILES and GRANULARITY parameters (as explained in the "Description" portion of this write-up). If not specified, the default extension size is 3 bytes.
INTERLEAVE=num IL I	Interleave factor for a NAMED or PHYSICAL volume. Acceptable values are 1 through 255 decimal. If not specified, the default value is 5. See the interleave discussion under "Description" in this section.
NAMED NA	The volume can store only named files; that is, the volume can hold many files, each of which can be accessed by its pathname. A diskette or hard disk surface are examples of devices that would be formatted for named files. If neither NAMED nor PHYSICAL is specified, the volume is formatted for the file type specified when you attached the device (with the ATTACHDEVICE command).
PHYSICAL PI P	The volume can be used only as a single, physical file. The GRANULARITY, FILES, and EXTENSIONSIZE parameters are not meaningful when PHYSICAL is specified for the volume. If neither NAMED nor PHYSICAL is specified, the volume is formatted for the file type specified when you attached the device (with the ATTACHDEVICE command).

BADTRACKFILE
BTFILE
BT

Names the file containing the bad track/sector information to be written to the volume. When this parameter is entered, the bad track/sector information area of the volume is read and user supplied information is merged with it and written to the disk before the volume is formatted. This parameter allows manufacturer's bad track information to be entered before actually formatting the disk. See the badtrack information discussion under "Description" in this section.

BTONLY

Identical to BADTRACKFILE, however, the rest of the volume is not formatted after the bad track/sector information is written.

OVERWRITE
OW
O

Indicates whether to overwrite the information existing on the volume with the bad track/sector information you provide. This parameter is only meaningful if the BADTRACKFILE option has been entered. If no option is entered and BADTRACKFILE has been given, the default option is merge. This means that FORMAT combines the bad track/sector information that you supply with the bad track/sector information already on the device.

QUERY
Q

Prompts the user for permission to format the volume. The Human Interface displays the following:

```
<volume name>, FORMAT?
```

If the user replies with a 'Y', 'y', 'R', or 'r', then the volume is formatted. Any other response is considered by the Human Interface to be a 'no'.

BOOTSTRAP
BS

Writes the second stage of the Bootstrap Loader onto track 0 without formatting the volume. When this parameter is specified, all other parameters are ignored.

RESERVE R	Creates the special file R?SAVE at the end of a volume after the volume has been initialized. The volume label file and the fnode file are then copied to R?SAVE. Later this file can be used in conjunction with the DISKVERIFY utility to back up the fnodes file on the volume. R?SAVE is not updated when files are altered. You update the R?SAVE file by using DISKVERIFY or by specifying backup in the SHUTDOWN command (discussed later in this chapter).

DESCRIPTION

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

Once a volume is formatted, its name becomes a volume identifier when you display any directory of the volume, and the name appears in the directory's heading.

VOLUME NAME

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

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

FNODES

The number of fnodes on a volume defines the number of files that can exist on the volume. You can specify the number of fnodes reserved for user files with the FILES parameter. Each fnode is a data structure that contains information about a file. Each time you create a file on the volume, the operating system records information about the file in an unused fnode. Later, it uses the fnode to determine the location of the file on the volume. You can enter the MAPSTART option to locate fnodes anywhere on a volume. If no MAPSTART option is entered, the operating system puts the fnodes in the center of the volume.

INTERNAL FILES

When you format a named volume, FORMAT creates six or seven (depending on whether the RESERVE option is set) internal system files. It names four (or five) of these files and lists their names in the root directory of the volume. The remaining two files are not listed in the root directory. Unless you specify the INVISIBLE option, none of these files appears when the DIR command is invoked. The files are:

File	Description
R?SPACEMAP	Volume free space map
R?FNODEMAP	Free fnodes map
R?BADBLOCKMAP	Bad blocks map
R?VOLUMELABEL	Volume label
R?SAVE	Save area for fnodes and volume label

The operating system grants the user WORLD read access to these files. The Save area file is only created if the RESERVE option is set. The volume label file is a special file occupying the first 3328 bytes of the volume. Refer to the *Extended iRMX II Disk Verification Utility Reference Manual* for more information about these files.

ROOT DIRECTORY

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

EXTENSION DATA

Each fnode contains a field that stores extension data for its associated file. An operating system extension can access and modify this extension data by invoking the AGETEXTENSION$DATA and A$SET$EXTENSION$DATA system calls (refer to the *Extended iRMX II Basic I/O System Calls Reference Manual* for more information). When you format a volume, you can use the EXTENSIONSIZE parameter to set the size of the extension data field in each fnode. Although you can specify any size from 0 to 255 bytes, the Human Interface requires all fnodes to have at least 3 bytes of extension data.

BADTRACK INFORMATION

It is possible to format only the bad track area of the disk and to write the bad track/sector information to this area from a file. Using the BADTRACKFILE parameter allows you to designate a file that will contain the bad track/sector information that is written to the volume. The existing bad track/sector information is read and user supplied information is merged with it and written to the area before the disk is formatted. If you specify the OVERWRITE parameter, any bad track/sector information existing on the volume is overwritten with the bad track/sector information you provide. The bad track information in the file must be in the format:

cylinder number head number sector number <CR> <LF>

where

cylinder number	the cylinder number of the bad track/sector
head number	head number of the bad track/sector
sector number	the number of the bad sector on the track indicated. On devices which only support bad track information, this value must be set to zero.

Remember that information entered in the badtrack file is not checked for validity, and only the first 255 triplets are used. The triplets may be separated by blanks, commas, carriage returns, or line feeds.

MAP FILES

If you have specified a map-files location (either implied or explicit) in an area which has a bad-track or for which an alternate was assigned, FORMAT allocates these files to the nearest available area, and then asks for permission to move the files in one of the following ways:

```
Map files located on a track assigned an alternate

Map files located on a bad track
```

A response of Y causes the files to be relocated and the following message to be displayed:

```
map start relocated to <hex-location>
```

This means you do not have to compute the location of the maps.

VOLUME GRANULARITY

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

RELATIONSHIP BETWEEN FILES, GRANULARITY, and EXTENSIONSIZE

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

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

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

INTERLEAVE FACTOR

The interleave factor applies to volumes formatted either for NAMED or PHYSICAL files. The interleave factor specifies the logical sector sequence. If the consecutively-accessed sectors of a disk are staggered (that is, if they are not consecutive physical sectors), disk access time can decrease considerably. The reason for this decrease is that although a controller cannot read a sector and issue another read command in the time it takes for the next sector to be positioned under the head, the controller can perform this operation in less time than it takes for the disk to revolve once. Therefore, if the consecutively-accessed sectors are staggered correctly, the next accessed sector will be positioned under the read head just as the controller becomes ready to read it.

The amount of staggering is called the interleave factor. An interleave factor of two means that as the disk rotates, the controller consecutively accesses every second sector. An interleave factor of five means that the controller consecutively accesses every fifth sector. The following diagram illustrates how a controller accesses sectors on a 12-sector disk with an interleave factor of two.

Sector Number	Access Number	Rotation Number
Sector 0	0	1
Sector 1	6	2
Sector 2	1	1
Sector 3	7	2
Sector 4	2	1
Sector 5	8	2
Sector 6	3	1
Sector 7	9	2
Sector 8	4	1
Sector 9	10	2
Sector 10	5	1
Sector 11	11	2

The interleave factor also implies the number of disk rotations necessary to access all the sectors on a given track in order. Thus from the previous diagram you can see that an interleave factor of two implies that it takes two rotations of the disk to access all the sectors on a track.

WHEN THE INTERLEAVE FACTOR IS IMPORTANT

The interleave factor is important when large transfers of consecutive data take place at speeds that approach the maximum transfer rate of the disk. This type of transfer occurs in the following cases:

- When you bootstrap load the operating system from disk.
- When you use the Application Loader to load an application program from disk.
- When you invoke programs that perform large transfers of consecutive data, such as the Human Interface COPY command.

HOW TO SELECT AN INTERLEAVE FACTOR

Suitable interleave factors depend on the turnaround time of the software that controls the I/O operations; that is, the time between reading a sector and becoming ready to read the next sector. In the cases listed in the previous paragraph, the turnaround time between sector accesses is different. Therefore the ideal interleave factors could be different. The differences are

- The Bootstrap Loader instructs the disk controller to read one sector at a time. Thus, the turnaround time depends on the execution overhead of the Bootstrap Loader and is comparatively long. A large interleave factor is optimal for flexible disks that you use with the Bootstrap Loader. For hard disks however, the Bootstrap Loader has no effect on the turnaround time because revolution speed is so great that more than one disk revolution occurs between sector reads.
- The Application Loader reads several sectors at a time into its internal buffer. Then it takes a relatively long time to process the object records in this buffer. The ideal interleave factor here is one that optimizes for the object record processing time between disk accesses. For flexible diskettes, this interleave factor is somewhat smaller than that for the Bootstrap Loader. However, hard disks, as in the previous paragraph, are not affected by the Application Loader.
- Applications which transfer large amounts of consecutive data (such as the COPY command) can initiate data transfers involving many sequential sectors. Thus the controller accesses sectors on a given track as fast as possible. Here, the ideal interleave factor is one that optimizes for the turnaround speed of the disk controller.

The ideal interleave factor depends heavily on the application. However, because the revolution speed of hard disks is so high, you should format them with interleave factors that are optimized for the turnaround speed of the disk controller.

The value to use for flexible diskettes depends on how you are going to use the diskettes. For flexible diskettes that contain bootstrap-loadable information (system disks), you should select an interleave factor that optimizes for Bootstrap Loader performance. This ensures that the bootstrap loading process completes in a reasonable amount of time, despite using a device that is relatively slow-turning. For non-system diskettes that contain loadable files (such as Human Interface commands), select an interleave factor that optimizes for Application Loader performance. Otherwise, select a value that optimizes for copying.

If you do not know the optimal value for an interleave factor, it is better to specify an interleave factor that is too large than one that is too small. An interleave factor that is slightly larger than optimal causes the disk to move only an extra sector or two before reaching the correct sector. However, an interleave factor that is less than optimal causes the disk to make nearly a complete revolution before reaching the sector.

OUTPUT DISPLAY

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

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

TTTTTTTTTTTTTTTTTT...

volume formatted
```

where

<number>	Decimal number as specified in the command (or the default).
<k-number>	Volume size in K (1024-byte units) or M (1048576-byte units). FORMAT displays the volume size in Kbyte units unless the size is greater than 25 Mbytes.

For named volumes:

```
volume (<volume name>) will be formatted as a NAMED volume
    granularity     = <number>          map start = <block-number>
    interleave      = <number>          sides     = <sides>
    files           = <number>          density   = <density>
    extensionsize   = <number>          disk size = <d-size>
    save area reserved = <yes/no>
    bad track/sector information written = <yes/no>
    volume size     = <k-number> K (or <M-number> M)

volume formatted
```

where

<volume name>	Volume name specified in the FORMAT command.
<number>	Decimal number as specified in the command (or the default).
<block-number>	Volume block number where the fnodes file, bit map files, and the root directory start.
<k-number>	Volume size in K (1024-byte units) or M (1048576-byte units). FORMAT displays the volume size in Kbyte units unless the size is greater than 25 Mbytes.
<sides>	Number of sides of the volume that will be formatted (1 or 2). This field is displayed only for flexible diskettes in which FORMAT can recognize this characteristic.
<density>	Density at which the volume will be formatted (single or double). This field is displayed only for flexible diskettes in which FORMAT can recognize this characteristic.
<d-size>	Size of the volume (8 or 5.25). This field is displayed only for flexible diskettes in which FORMAT can recognize this characteristic.
<yes/no>	Yes or no. Indicates whether the R?SAVE file has been reserved for backing up the label and the fnode file. This response also specifies whether or not bad track/sector information is to be written.

While the storage device is being formatted, FORMAT displays the letter "T" on the console for every 100 tracks formatted. For example, if you see three T's on the screen, the operating system has finished formatting at least 300 tracks. Displaying the T's on the screen is useful when you format large capacity disks. A continuous stream of T's lets you know that the system hasn't failed during the FORMAT operation.

If the BOOTSTRAP parameter is specified, the message:

```
Bootstrap Loader written
```

is displayed upon completion instead of volume formatted.

When the BTONLY parameter is specified, the message:

```
Bad Track/Sector Block written
```

is displayed instead of volume formatted.

If the EIOALT$ASSIGNED error code is returned by a driver when formatting a track, the track number is entered into a table and displayed as follows when volume format is complete.

```
The following tracks were assigned an alternate:

cyl    hd    cyl    hd      cyl    hd    cyl    hd    cyl    hd
cyl#   hd#   cyl#   hd#     cyl#   hd#   cyl#   hd#   cyl#   hd#
```

There should be an entry in this table for every BTI track specified, except those that reside in the alternate track area.

If the EIONO$SPARES error code is returned by a driver when formatting a track, this means that the number of alternate tracks reserved is exhausted. The sectors of that track are marked in the Bad Block Map File and entered in the Volume Space Map File as they were assigned. The track is entered into a table and displayed as follows when volume format is complete.

```
The following tracks were marked as bad:

cyl    hd    cyl    hd      cyl    hd    cyl    hd    cyl    hd
cyl#   hd#   cyl#   hd#     cyl#   hd#   cyl#   hd#   cyl#   hd#
```

where

cyl# The cylinder number in hexadecimal

hd# The head number in hexadecimal.

BOOTSTRAP LOADER AND THE FORMAT COMMAND

The Bootstrap Loader operates in three stages. The second stage must reside on track 0 of any iRMX-named volume. The second stage that is placed in track 0 by the iRMX I FORMAT command (for users upgrading from an iRMX I Operating System) cannot be used to bootload an iRMX II Operating System. Therefore, if you want to bootload the iRMX II Operating System from a named volume, be sure that the volume has been formatted using the iRMX II FORMAT command. The iRMX II second stage can be used to load iRMX I.

To avoid forcing you to reformat entire disks when the second stage of the Bootstrap Loader changes, you can specify the BOOTSTRAP parameter to write the second stage of the Bootstrap Loader onto track 0 without reformatting the rest of the volume.

To install the second stage of the Bootstrap Loader onto a named volume, invoke the FORMAT command, indicating the logical name of the appropriate volume and specifying the BOOTSTRAP parameter. When the BOOTSTRAP parameter is specified, any other parameters entered with the command are disregarded. FORMAT writes the second stage of the Bootstrap Loader onto track 0 without reformatting the volume.

CAUTION

If you fail to specify the BOOTSTRAP parameter, FORMAT will format the entire volume.

For example, assume that you have a disk formatted by the iRMX I version of the FORMAT command. You plan to use iRMX II.2 or iRMX II.3 versions of the Bootstrap Loader with some of the files on this disk. To avoid the time-consuming process of saving the disk, reformatting and restoring its contents, you can invoke the FORMAT command with the BOOTSTRAP switch. This copies the second stage of the Bootstrap Loader onto track 0 of the iRMX I disk.

Any of the following commands will copy the second stage of the iRMX II.2 Bootstrap Loader onto track 0 of the device that was attached with the logical name :f:.

```
-
-
-
```

When the FORMAT command has completed executing, track 0 of the volume contains the iRMX II.2 or iRMX II.3 version of the Bootstrap Loader's second stage. The remainder of the files on the volume are unaffected. (Notice, in the third example, the FILES, GRANULARITY, and FORCE switches are ignored because the BOOTSTRAP parameter has precedence over any other FORMAT parameter.)

ERROR MESSAGES

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

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

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

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

- `<logical name>, device is in use`

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

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

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

- `<number>, invalid number`

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

- `<logical-name>, map files do not fit`

  The volume is too small for the map files or the map start block is too high to allow room for the map files.

- `map files do not fit`

  Either the volume is too small for the map files, or the map start block is too high in disk storage memory to allow for the map files.

- `map files do not fit with save area`

  Either the volume is too small for both the map files and the save area, or the map start block is too high in disk storage memory to allow for the map files and the save area.

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

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

- `0085 : E$LIST, too many values`

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

- `<logical-name>: <exception code>: <exception name>`
  `unit status <unit status code> while writing block number`

  An I/O error occurred while writing the label, map files, or save area to a named file.

- `<logical-name>: <exception code>: <exception name>`
  `unit status <unit status code> while formatting track`

  An I/O error occurred while physically formatting the volume. If an EIOALT$ASSIGNED error code has been returned, you can consider this message as a warning.

- `<volume name>, volume name is too long`

  The volume name must not be longer than six characters.

- `Track zero bad, cannot write`

  The volume label track (track zero) is marked in the Bad Block Map.

- `cannot relocate`

  A warning message displayed when the map files are located on a sector or sectors which have been assigned an alternate and a suitable location on the disk cannot be found.

- `cannot relocate...aborting`

  An error message displayed when the map files are located on a sector or sectors which have been marked in the Bad Block Map and an alternate location cannot be found.

- `Save file located on a bad track, cannot write`

  The save area is located on a sector or sectors which have been marked in the Bad Block Map.

- `<file name>, cannot open bad track/sector information file`
  `<file name>, <exception code>`

  The file containing the bad track/sector information cannot be opened for reading.

- `too many bad track/sector information entries`

  The file containing the bad track/sector information has too many entries or the combination of the file entries and the information existing on the volume cannot be merged.

- `<file name>, illegal bad track/sector information`

  The file containing the bad track/sector information has the wrong format.

- `BADTRACKFILE option missing, cannot replace Bad Track/Sector Information Block`

  The OVERWRITE option was entered without the BADTRACKFILE parameter.

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

DESCRIPTION

INITSTATUS displays at the user terminal the initialization status of all Human Interface terminals. Figure 4-7 illustrates the format of the INITSTATUS display.

```
TERMINAL      CONFIG    DEVICE   INIT     TERM    JOB      USER     USER     USER
DEVICE NAME   EXCEP     EXCEP    EXCEP    STATE    ID       ID      POOL     NAME

.T0.          0000      0000     0000     D-E      1         0     1,400K   rmx
.T1.          0000      0000     0000     SLE      2     65535     1,400K   rmx
.T3.          0000      0002              D--
.T4.          0021                        D--
```

Figure 4-7. INITSTATUS Display

The columns listed in Figure 4-7 contain the following information.

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

DEVICE EXCEP	Hexadecimal condition code that the Human Interface received when it originally attached the terminal as a physical device.
INIT EXCEP	Condition code that the Human Interface received when it created a job for the interactive session.
TERM STATE	Three characters that indicate the current state of the terminal. The first character can be either: D The terminal is a dynamic logon terminal. S The terminal is a static logon terminal. The second character can be either: L The terminal is locked (refer to the LOCK command and UNLOCK commands later in this chapter). - The terminal is unlocked. The third character can be either: E The Human Interface created the interactive job associated with this terminal and the job exists. - The interactive job does not exist.
JOB ID	A sequential number that the Human Interface assigns to the interactive job during initialization. You must specify this number as the parameter in the JOBDELETE command in order to delete the corresponding interactive job.
USER ID	User ID associated with the interactive job. This ID is the identification of the user that the Human Interface associates with the job when the user begins a Human Interface session.

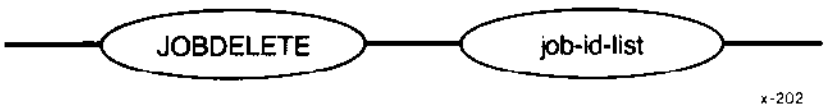
USER POOL	The maximum size of the memory partition that is associated with the interactive job.
USER NAME	The logon name of the user who is accessing this terminal.

ERROR MESSAGE

- not a multi-user system

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

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



where

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

DESCRIPTION

The JOBDELETE command allows users to delete interactive jobs. Deleting an interactive job causes the Human Interface to terminate the corresponding user session. JOBDELETE cannot be used to delete background jobs. To delete a background job, you must use the CLI command KILL (see Chapter 3).

When JOBDELETE attempts to delete a job, it first attempts to delete the job's offspring jobs (for example, a SUBMIT file or a program invoked as a result of an RQE$CREATE$IO$JOB system call). It deletes multiple levels of offspring jobs. However, JOBDELETE cannot delete any interactive job (or offspring) that contains extension objects. Refer to the *Extended iRMX II Nucleus User's Guide* for more information about deleting jobs containing extension objects.

Normally, when a user's interactive job is deleted, the Human Interface logs the user off the system and issues the prompts for a new user to log on (unless the user is at a static logon terminal, in which case, the Human Interface automatically recreates the interactive job, thus restarting the user session). However, if the LOCK command (described later in this chapter) has been specified for the user's terminal, the Human Interface does not prompt for logon or recreate interactive jobs after a JOBDELETE command. Therefore, the system manager can use the combination of LOCK and JOBDELETE to remove users from the system before a system shutdown.

Unless you delete your own interactive job, JOBDELETE displays the following message at the user terminal (:CO:) as it deletes each job:

```
<job-ID>, deleted
```

where <job-ID> is the identifier of the deleted job. If you delete your own interactive job, you will see the logon prompt (for dynamic logon terminals) or your interactive job will be restarted (static logon terminals).

ERROR MESSAGES

- `<job-ID>, does not exist`

  The interactive job associated with the identifier <job-ID> does not exist. It has already been deleted or never existed.

- `<job-ID>, invalid job id`

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

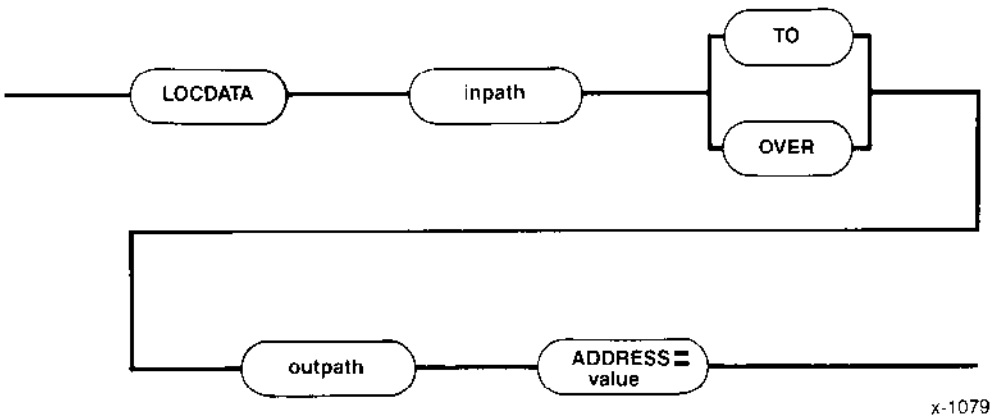
- `<job-ID>, job does not belong to you`

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

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

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

This command, in conjunction with the Human Interface ADDLOC command, integrates the image of a data stream (such as a RAM disk) into an existing application system boot file. LOCDATA transforms a data stream (the physically attached RAM disk) into a located data file (that is, a file that identifies the absolute address at which the Bootstrap Loader must load the RAM disk). Using the located data file, ADDLOC creates a new boot file containing a version of the application system that includes the RAM disk image. The format of this command is as follows:



INPUT PARAMETERS

inpath — The logical name of the physically attached RAM disk. Multiple or wild-card pathnames are not allowed.

ADDRESS=value — The address at which the Bootstrap Loader is to load the data stream (for example, the address of a RAM disk). The address must specify a WORD boundary. Be careful when assigning this address that you do not overlay any part of the system or the third stage of the Bootstrap Loader.

You can specify a radix character of "O" or "H" at the end of the value to indicate octal or hexadecimal, respectively. If the radix character is omitted, decimal is the default.

OUTPUT PARAMETERS

TO — Writes the processed output to a named file. If the specified file already exists, LOCDATA displays the following message:

```
<pathname>, already exists, OVERWRITE?
```

To overwrite the existing file, enter Y, y, R, or r. If you do not wish to overwrite the existing file, enter E, e, N, or n. LOCDATA then exits without processing the data.

OVER	Overwrites the existing output file. If the specified file does not already exist, LOCDATA creates it.
outpath	Pathname of the file to receive the output of LOCDATA. Multiple or wild-card pathnames are not allowed.

DESCRIPTION

The iRMX II Operating System supports the use of a RAM disk, an area of memory that is treated as a secondary storage device. To use the RAM disk feature you must configure a system with an area of Random Access Memory dedicated to the RAM disk. When the system boots you can attach the RAM disk memory to your system, format it, and move data into and out of the RAM disk just as you would with any other disk device. If you use the RAM disk to store part of the application system (for instance, the Human Interface commands), the stored data must be available in the RAM disk area when the system boots. This data can not be copied into the RAM disk until you have configured the application system into a bootable file. (The RAM disk area does not exist until it is defined through the configuration process.) Therefore, there must be some method of integrating a copy of a RAM disk data structure into an existing application system boot file.

In the iRMX II Operating System, this mechanism is provided by the Human Interface LOCDATA and ADDLOC commands. Using the address assigned to the RAM disk during the configuration process, LOCDATA creates a "located" data file containing the image of the RAM disk. (A "located" file is a file that specifies the starting address at which it is to be loaded by the Bootstrap Loader.) ADDLOC uses the application system bootfile and the file output by LOCDATA to create a new bootable version of the application system. The new version includes a copy of the RAM disk data structure. When this new file is booted, the RAM disk data structure is loaded into memory in the area defined for the RAM disk through the configuration process.

When invoking LOCDATA, the first parameter you supply must be the logical name of a physically attached RAM disk. (You should have already formatted the RAM disk and created the necessary data structures before detaching and reattaching it as a physical device.) The second parameter must be the pathname to be assigned to the located data file. The third parameter must specify the starting address where the Bootstrap Loader is to load the RAM disk. This address must be the same starting address assigned to the RAM disk when you configured the application system.

After processing the data, LOCDATA displays one of the following messages:

```
<inpath>, located TO <outpath>
```

```
<inpath>, located OVER <outpath>
```

Creating an Application System that Includes a Bootable RAM Disk

To create an application system that contains a RAM disk that is initialized through the Bootstrap Loader, perform the following steps:

1. Configure a version of the operating system that includes a RAM disk. (Refer to *Extended iRMX II Interactive Configuration Utility Reference Manual* for more information.) Make a special note of the starting address you specify for the device.
2. Bootstrap load this new version of the operating system.
3. Attach the RAM disk as a named device. For example:

4. Format the RAM disk as a named file. For example:

5. Create the appropriate data structure on the RAM disk and copy the files that you need, such as the Human Interface commands, to the appropriate directory on the RAM disk. An error message occurs if you run out of room in the RAM disk.
6. Detach the RAM disk. For example:

7. Attach the RAM disk as a physical device. For example:

   This allows you to access all the data in the device, including the formatting information.
8. Use the LOCDATA command to process the information from the RAM disk and place the output in another file. Use the RAM disk starting address (specified during configuration) as the value for the ADDRESS parameter. Thus, if you configured your RAM disk to have a base address of 0100000H, the following example applies:

9. Use the ADDLOC command to add the processed output (in this case, the file COMMANDS) to the file that contains the bootstrap loadable version of the operating system. For example:

   The processed output file of the LOCDATA command (in the example, the file COMMANDS) will be combined with a bootloadable file (in the example, the file RMX86.286) to produce a new bootloadable file (in the example, RAMDISK.286). The ADDLOC process will generate a print file (RAMDISK.MPA).

10. Create a Bootstrap Loader third stage for the newly created bootable file. For example:

Now, whenever you bootstrap load this new version of the operating system, the RAM disk will contain the commands and files copied to it during Step 5.

ERROR MESSAGES

The error messages listed below are in addition to the standard Human Interface error messages.

- `<pathname>, is a keyword not a file name`

  One of the pathnames you specified was a keyword not a file.

- `LOCDATA, one input file only`

  LOCDATA requires exactly one input file. You specified more than one input file.

- `LOCDATA, one output file only`

  LOCDATA requires exactly one output file. You specified more than one output file.

- `AFTER, is an illegal preposition for LOCDATA`

  The AFTER preposition, which you entered in your invocation line, is not a legal LOCDATA preposition.

- `<string>, illegal preposition`

  The preposition entered in your invocation line is not a legal LOCDATA preposition.

- `LOCDATA, ADDRESS parameter is missing`

  You failed to specify the ADDRESS parameter in the invocation line.

- `LOCDATA address value is missing`

  You failed to enter the address value in the invocation line.

- `LOCDATA, no more than one address value`

  You entered more than one address value in the invocation line

- `LOCDATA, illegal address value`

  The address value you specified is not within the range of 0 to 0FFFFFFH.

- `<string>, unrecognized control`

  You specified an unrecognized control in the invocation line.

- `<pathname>, output file same as input file`

  You have entered the same name for both the input and output file. LOCDATA does not allow this.

- `<pathname>, write error`

  A system error caused an incorrect number of bytes to be written to the output file. Retry the command.

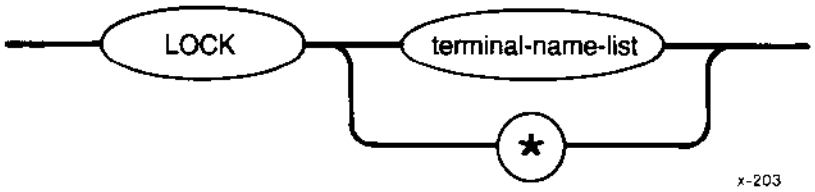
- `<pathname>, physical address exceeded 16M bytes`

  The base address added to the size of the input file you specified exceeds 16M bytes.

- `<pathname>, read error`

  A system error caused an incorrect number of bytes to be read from the input file. Retry the command.

This command prevents the Human Interface from recreating an interactive job or issuing a logon prompt for a particular terminal once the interactive job that was active on that terminal has been deleted. As a result, users cannot access the Human Interface through that terminal. This process is called locking the terminal. The system manager can use this command to lock any terminal. Other users can lock only those terminals whose interactive jobs have the same user ID that they have. The format of this command is as follows:



where

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

DESCRIPTION

The system manager can use the LOCK command in conjunction with the JOBDELETE command either to selectively delete users from the system or to shut down the entire system. LOCK prevents users from logging onto a dynamic logon terminal (or the Human Interface from recreating an interactive job on a static logon terminal) once the previous interactive job has been deleted. Interactive jobs can be deleted in any of the following ways:

- By entering the JOBDELETE command (described earlier in this chapter)
- By entering the LOGOFF command (described in Chapter 3)

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

```
locked
<terminal-name>, locked
```

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

However, if a terminal is currently displaying a logon prompt, LOCK cannot prevent a user from trying to logon. If that logon attempt fails the terminal will be locked for all subsequent logon activities.

ERROR MESSAGES

- `lock not allowed`

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

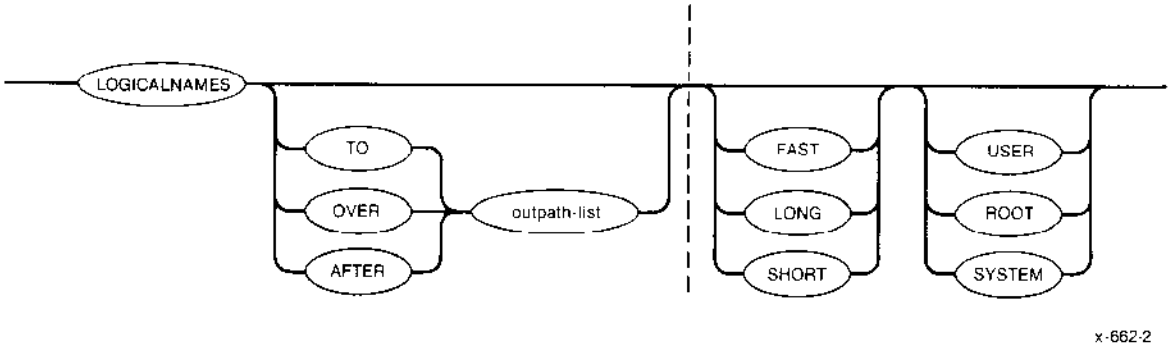
- `<terminal-name>, not found`

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

- `not a multi-user system`

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

This command lists all the current logical names available to the user. The format of the command is as follows:



INPUT PARAMETERS

FAST	Lists the logical names in a system without providing any additional information beyond the name itself. FAST is the default parameter.
SHORT	Lists all the logical names with the following additional information: type of logical name, the physical device name, owner of the logical name, and the current connections to the file or device.
LONG	Like the SHORT parameter, but also adds the complete pathname associated with a logical name.
ROOT	If ROOT is specified with the LONG parameter, the full pathname (back to the root device) associated with the logical name is displayed.
USER	Displays all the logical names associated with the current user.
SYSTEM	Displays the logical names of system defined files and devices.

OUTPUT PARAMETERS

TO — Writes the listed logical names to the named output files. The specified output file or files should not already exist. If they do, the following message is displayed:

```
<pathname>, already exists, OVERWRITE?
```

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

OVER — Writes the input files over (replaces) the existing output files on a one-for-one basis, regardless of file size. If an output file does not already exist, its corresponding input file is written to a new file with the corresponding output file name.

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 — The pathname of the file to receive the output of the command.

DESCRIPTION

The following is an example of the listing you get when you invoke the command with a FAST control (FAST is also the default parameter):

```
-LOGICALNAMES FAST <CR>
User Logical Names:

PROG     TERM     $        CI       CO
HOME

System Logical Names:

LANG     WORK     UTILS    SYSTEM   CONFIG
*SD      *LP      *STREAM  *BB
```

When an asterisk precedes a name, the logical name refers to a logical device.

The following example shows the output listing when you use the SHORT parameter:

```
-LOGICALNAMES SHORT <CR>
User Logical Names:
  name        type        fdr      con     dev name      owner
$              dir         NAM       3      smd0          WORLD
CI            file         PHY       5      T1
CO            file         PHY       5      T1
HOME           dir         NAM       3      smd0          WORLD
PROG           dir         NAM       2      smd0          WORLD
TERM          file         PHY       5      T1
REMOTE1       file         REM       1      server1       WORLD

System Logical Names:
  name        type        fdr      con     dev name      owner
SYSTEM         dir         NAM       1      smd0           #0
WORK           dir         NAM       1      smd0          WORLD
SD            ldev         NAM       1      smd0           #0
BB            ldev         PHY              BB             #0
STREAM        ldev         STR       1      STREAM         #0
REMSYS        ldev         REM       0      server1       WORLD
```

In the listing, type refers to the kind of logical name: file, directory, map (system file), or logical device (ldev). Fdr (file driver) indicates whether the connection is to a named, physical, stream, or remote file. The number of connections which a file or device has is under the con heading. The dev name heading shows the physical device associated with the logical name. In the case of a directory or file, the name shows on what device the file or directory exists. The originator of the connection to the logical name is shown under the owner heading.

The use of the LONG parameter produces the same type of listing as the SHORT parameter with the addition of the complete pathname of the logical name. Following is an example of the LONG listing:

```
-LOGICALNAMES LONG <CR>

User Logical Names:

  name       type       fdr      con      dev name   owner     pathname
$             dir        NAM       3       smd0       WORLD    :SD:user/world
CI           file        PHY       5       T1                  :CI:
CO           file        PHY       5       T1                  :CO:
HOME          dir        NAM       3       smd0       WORLD    :SD:user/world
PROG          dir        NAM       2       smd0       WORLD    :$:prog
TERM         file        PHY       5       T1                  :TERM:

System Logical Names:

  name       type       fdr      con      dev name   owner     pathname
SYSTEM        dir        NAM       1       smd0       #0       :SD:sys286
WORK          dir        NAM       1       smd0       WORLD    :SD:work
SD           ldev        NAM       1       smd0       #0       :SD:
BB           ldev        PHY               BB         #0       :BB:
STREAM       ldev        STR       1       STREAM     #0       :STREAM:
REMSYS       ldev        REM       0       tahoe2     WORLD    :TAHOE:
```

The ROOT parameter, in conjunction with the LONG parameter, produces the same type of listing as the LONG parameter except that the pathname starts at the root directory. Thus, you get a complete description of the pathname associated with the logical name.

If the pathname has ellipses before it (.../user/dir1/dir2/dir3/filename), LOGICALNAMES truncated the pathname because it was too long to fit in its column. The pathname only shows the last elements of the pathname which describes a file or directory.

This command logs the user off a dynamic logon terminal. For users operating under the Human Interface supplied CLI, this is a CLI command. LOGOFF is only used as a Human Interface command if your system has its own custom CLI. For a complete description of this command, see Chapter 3.

This command displays the memory currently allocated to the user and the total system memory available to the user.

DESCRIPTION

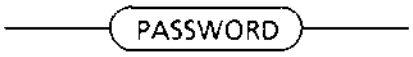
This command requires no parameters. The following is an example of the listing produced by this command:

```
-MEMORY <CR>
User Private Memory (pool minimum) :   300 k Bytes
Available Memory (Private + Shared): 1.545 M Bytes
```

where:

User Private Memory	the amount of memory currently allocated to the user
Available Memory	the amount of memory available for the user, that is, the private memory and the amount of memory the user job can borrow from its parents. For example, if your private memory is 300K bytes and your total memory is 1.545M bytes, as shown above, you can still borrow 1.245M bytes.

PASSWORD performs a number of actions depending on the user ID of the operator invoking the command. For the system manager (user ID 0), this command can add a user to the User Definition File (UDF), delete a user from the UDF, read the UDF, or change a logon password. For non-system managers (user ID greater than 0), this command enables the user to change his or her logon password. The UDF must have read access rights for WORLD. The format of the command is as follows:



F-0665

DESCRIPTION

If you are not the system manager, you can invoke the PASSWORD command to change the password you enter when logging onto the Human Interface from a dynamic logon terminal. However, if your system's UDF resides on a remote system, the system manager must change your password for you (or allow you to do it on the system manager's behalf).

After you invoke the command, the following messages appear:

```
Enter your user name -
Enter the old password -
```

In response, enter your logon name and your current password (the one you want to change). For security reasons, the password you enter is not echoed on the screen. The command then prompts you for the new password with the following message:

```
Enter the new password -
```

In response, enter your new password. The new password must be no longer than eight characters (more will be ignored). To minimize the effect of typing errors, the command asks you to repeat the password, as follows:

```
Repeat the new password -
```

In response, again type your new password. After confirming that both entries of the new password are identical, the command associates that new password with your logon name and displays the messages:

```
Password change successful
Updating the master UDF ......... Done
```

The next time you log onto the system, you must use this new password. Continue using it until you change your password again with the PASSWORD command.

If you are the system manager (user ID 0), the PASSWORD command performs a varicty of functions, including maintaining the User Definition File (UDF).

The UDF contains the logon name, user ID, and password of all users who can access the Human Interface via a dynamic logon terminal. Because this file is also used to validate user access to an OpenNET environment, the file requires a nonstandard format (one that precludes using an ordinary text editor to maintain the file). In addition, the passwords listed in the UDF are encrypted to prevent unauthorized access by those who happen to see a listing of the UDF. The PASSWORD command is the sole mechanism for maintaining the UDF, and only the system manager can use this mechanism. PASSWORD adheres to the nonstandard formatting of the file, and it automatically encrypts the passwords it adds or changes.

As the system manager, when you invoke the PASSWORD command, the following menu appears on the terminal screen:

```
The following commands are available:

A - Add a user
D - Delete a user
L - List the UDF
C - Change password
Q - Quit
E - Exit

Enter the command:
```

To perform any of the operations listed on the menu, enter the letter associated with that operation. For example, to add a new user to the system, type the letter "A" and press carriage return.

The following sections explain the operations listed in the PASSWORD menu.

Adding a User to the UDF

Choose the A option to add a new user to the UDF. PASSWORD responds by prompting you to enter information about the new user. The prompts (and valid answers) are as follows:

```
Enter the user name -
```

Enter the logon name of the new user. This name must be three to eight characters long. If you respond with more than eight characters, the command ignores the extra characters. You must respond to this prompt.

```
Enter the new password -
```

Enter the password for the new user. This password must be eight characters or less (additional characters are ignored). If you enter a carriage return only, the new user will not have a password initially.

Entering the characters:

prevents the user from logging onto the system.

```
Repeat the new password -
```

Enter the new password again. This double checking validates the password and ensures that you spelled it correctly. PASSWORD returns an error message if the two passwords don't match and reprompts for the new password. This continues until you enter the new password correctly twice.

```
Enter the user ID -
```

Enter a user ID to associate with this user. The user ID must be a decimal number in the range of 0 to 65535. If you enter a carriage return only in response to this prompt, PASSWORD assigns the next higher user ID that is not in use and responds with:

```
Assigned user ID of <ID>
```

If there are no unique user IDs available or the ID you enter is not unique, PASSWORD displays the warning:

```
Warning - Not a unique user ID
```

Assigning a user ID that is not unique can cause problems in a network environment.

Entering any other value causes the command to display an error message and repeat the prompt. This will continue until you enter a valid user ID, a carriage return, or a "Q" (to abort this session of adding a user).

```
Enter the group ID -
```

If your system is part of the OpenNET and includes XENIX workstations, refer to the XENIX documentation for more information on the appropriate responses to this prompt. Otherwise, enter a second user ID which will be added to this user's iRMX user object. If neither of the user ID's are 65535 (WORLD), the HI will automatically add 65535 (a third ID) to the user object when the user logs on.

```
Enter the comment -
```

Respond to this prompt with a carriage return unless your iRMX II system is an OpenNET workstation. The iRMX II Operating System does not use this field. Refer to the XENIX documentation for more information.

```
Enter the default Xenix directory -
```

Respond to this prompt with a carriage return unless your iRMX II system is an OpenNET workstation. For OpenNET workstations, answer this prompt with the complete pathname of the new user's XENIX home directory.

```
Enter the default Xenix shell -
```

Respond to this prompt with a carriage return unless your iRMX II system is an OpenNET workstation. For OpenNET workstations, supply the new user's default shell. Refer to the XENIX documentation for more information.

Once you have responded to all the prompts, PASSWORD summarizes and displays all of your answers in the following way:

```
User name = <answer to "Enter the user name" prompt>
User ID = <answer to "Enter the user ID" prompt>
Group ID = <answer to "Enter the group ID" prompt>
Comment = <answer to "Enter the comment" prompt>
Default Xenix directory = <answer to "Enter default XENIX directory
                          prompt
Default Xenix Shell = <answer to "Enter the default XENIX shell
                      prompt>

Do you want to add this user to UDF?
```

Respond with a "Y" or "y" to add the user. If you respond to this prompt with any character other than "Y" or "y", PASSWORD disregards your previous input and returns to the initial menu.

Entering a "Y" or "y" causes PASSWORD to update the copy of the User Definition File (UDF) it maintains in memory, (the permanent copy will be updated when you invoke the Exit (E) command) and displays this message:

```
Do you want to create the user directories?
```

A response of "No" means the system manager must manually create the user's home directories. In this case, PASSWORD will create the user configuration file :CONFIG:user/<username> unless it already exists.

A response of "Yes" creates directories, copies the alias.csd and R?LOGON files from the :config:default directory, and creates an empty R?LOGOFF file in the new user's PROG directory. After the files are created, you are prompted for the pathname of the initial program as follows:

```
Initial-program pathname =
```

If you are using the standard CLI, enter a carriage return. If you are not using the standard CLI, enter the full pathname of the CLI you are using. After adding the new user, PASSWORD responds with the following message:

```
Default Initial Program is RMX286 HI CLI
Added user <user name>
```

Refer to the *Extended iRMX II Interactive Configuration Utility Reference Manual* for more information about configuration files.

Deleting a User from the UDF

Choose the D option to delete a user from the UDF. PASSWORD responds by prompting you to enter information, as follows:

```
Enter the user name :
```

Enter the logon name of the user to be deleted. If the name you enter is currently listed in the UDF, PASSWORD deletes the entry from the copy of the UDF it maintains in memory and responds with the following message:

```
Deleted user <logon name>
```

The permanent copy of the UDF will be updated when you invoke the Exit (E) command.

Listing the Contents of the UDF

Choose the L option to list the contents of the User Definition file. PASSWORD responds by displaying the following information:

```
<logon name>:<password>:<user id>:<group id>:<comment>:<dir>:<shell>
<logon name>:<password>:<user id>:<group id>:<comment>:<dir>:<shell>
      .          .          .          .          .          .          .
      .          .          .          .          .          .          .
      .          .          .          .          .          .          .
<logon name>:<password>:<user id>:<group id>:<comment>:<dir>:<shell>
```

where:

<logon name>	Logon name.
<password>	Encrypted password. No entry indicates that the user does not require a password to log onto the system. The characters "NO LOGIN" indicate that the user is prohibited from logging on.
<user id>	Decimal number representing the user ID. Values that represent special kinds of users include:

Value	User
0	System manager
65535	WORLD

<group id>	A second ID that can be implemented as a group convention. This second ID is added to the user's iRMX user object.
<comment>	Comment field (used only in OpenNET systems).
<dir>	XENIX directory (used only in OpenNET systems).
<shell>	XENIX shell (used only in OpenNET systems).

Changing Passwords

Choose the C option to change the logon password. PASSWORD responds by prompting you to enter information, as follows:

```
Enter your user name -
Enter the old password -
```

In response, enter the logon name and the current password (the one you want to change). If you do not type the correct old password, the following message will be displayed:

```
Old password is incorrect
```

and you will be prompted again to enter the old password.

When you have entered the old password correctly, the command prompts you for the new password with the following message:

```
Enter the new password -
```

In response, enter the new password. The new password must be no longer than eight characters. Entering the characters:

prevents the user from logging onto the system. To minimize the effect of typing errors, the command asks you to repeat the password, as follows:

```
Repeat the new password -
```

In response, again type the new password. If the passwords are not identical, PASSWORD returns the following error message:

```
Invalid password
```

and reprompts you for the new password. This process continues until you enter the new password correctly twice. After confirming that both entries of the new password are identical, the command associates the new password with the logon name and displays the following messages:

```
Password change successful
Updating the master UDF ..... Done
```

Quitting the PASSWORD Command

To abort the PASSWORD command without saving any of the changes you made during this session, choose the Q command. If you have made any changes that will be lost, PASSWORD responds with the following message:

```
Do you really want to quit without saving your changes?
```

If you want to abort the session and lose all of the changes you made, enter "Y" in response. Entering any other character returns you to the main menu without discarding your changes.

Exiting the PASSWORD Command

To leave the PASSWORD command and save all of the changes you made during this session, choose the E command. PASSWORD writes to the User Definition File all of the changes it previously held only in memory. Your screen then displays the Human Interface prompt.

ERROR MESSAGES

- `Cannot attach to the UDF`

  The operating system encountered an error, either when attempting to read the password you entered or when attempting to access the UDF.

- `Illegal name`

  The logon name you specified is invalid. The name must be between three and eight characters long, contain no embedded blanks, and contain no unprintable characters.

- `Invalid command`

  You entered an invalid command at the PASSWORD menu. The valid commands are A, D, L, C, Q, and E.

- `Invalid Password`

  Either the password you entered was longer than eight characters, or you made a typing error when you confirmed the password by entering it again.

- `Invalid response`

  Your response to a prompt was invalid. For example, you might have entered alphabetic characters when a numeric value was expected.

- `Maximum size of UDF reached`

  The User Definition File can grow to a maximum of 32K bytes. It has reached this limit, and no more new users can be added.

- `<Master/Local> UDF is not available`

  An error occurred while PASSWORD was attempting to attach the UDF. If your system is part of an iRMX-NET environment, the error occurred while attaching the remote master UDF. If your system is not part of an iRMX-NET environment, the error occurred while attaching the local UDF. In either case, PASSWORD did not change the UDF.

- `Old Password is incorrect`

  The password you entered did not match the password listed in the UDF.

- `UDF does not exist.`

  Your iRMX II system is not configured to support nonresident users. Therefore, the User Definition File does not exist.

- `UDF does not exist. Creating new UDF.`

  The User Definition File did not exist on your system before, because your system is not configured to support nonresident users. As the system manager, you can add a UDF. The PASSWORD command creates a UDF to contain your additions.

- `UDF is corrupted`

  The User Definition File has an invalid format that must be fixed. This might have been caused by editing the file with a text editor. To correct this problem, the system manager might need to delete the UDF (with the DELETE command) and use the PASSWORD command to rebuild it. A copy of the original iRMX II UDF is kept in :CONFIG:default/udf.

- `UDF is not available`

  The User Definition File can be written by only one user at a time. Someone else is using the PASSWORD command now and has exclusive write access to the UDF. Try again in a few seconds.

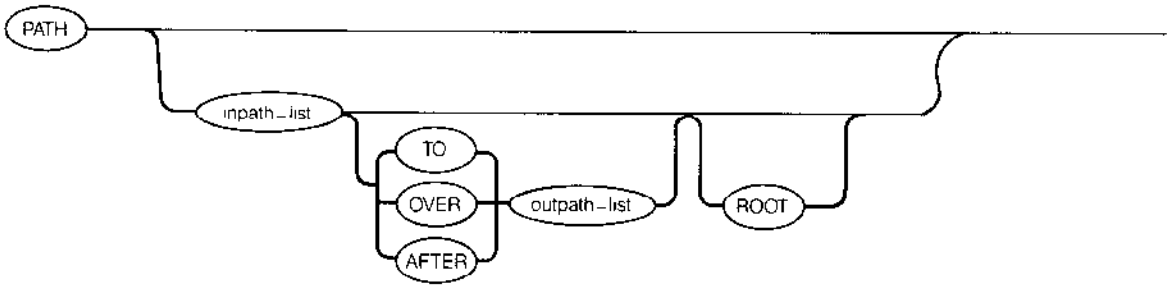
- `User <logon name> is already defined in the UDF`

  The user you attempted to add is already listed in the UDF.

- `User <logon name> is not defined in the UDF`

  The user you attempted to delete is not listed in the UDF.

This command lists the pathname of a data file or directory.



x-941

INPUT PARAMETERS

inpath-list — The list of files whose pathnames you want to know. The default inpath-list file directory is the current working directory (:$:).

ROOT — Specifies that the pathname should start from the root directory of whatever device holds the file or directory.

OUTPUT PARAMETERS

TO — Writes the pathnames of the input files to the specified output files. The specified output file or files should not already exist. If they do, PATH displays the following message:

```
<pathname>, already exists, OVERWRITE?
```

Enter Y, y, R, or r if you wish to write over the existing file. Enter any other character or a carriage return alone if you do not wish to overwrite the existing file. In the latter case, the PATH command will pass over the corresponding input file, and will attempt to write the pathname of the next input file to the corresponding output file.

If you specify multiple input files and a single output file, PATH appends the remaining input file pathnames to the end of the output file.

OVER	Writes the input file pathname over (replaces) the existing output files on a one-for-one basis, regardless of file size. If an output file does not already exist, the corresponding input file pathname is written to a new file with the corresponding output file name. If you specify multiple input files and a single output file, PATH appends the remaining input file pathnames to the end of the output file.
AFTER	Appends the input file pathname(s) to the current data in the existing output file or files. If the output file does not already exist, all listed input file pathnames will be concatenated into a new file with the listed output file name.
outpath-list	One or more pathnames for the output files.

DESCRIPTION

This command is useful for finding where you are located within the file structure. The command gives the following listing when it is invoked with no input file listing:

```
-PATH <CR>
:SD:user/world
```

This command displays an optional message on the console and waits for you to enter a carriage return.

F-0830

INPUT PARAMETERS

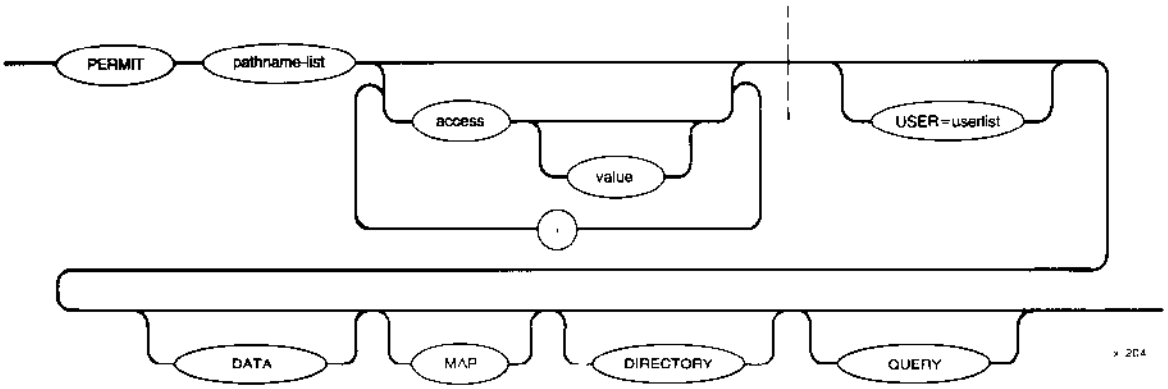
message-text — The text that appears on the console when the PAUSE command is executed.

DESCRIPTION

This command prompts the user's console with a message and waits for a carriage return. PAUSE works ideally when executed from within a SUBMIT file. PAUSE, however, aborts if you attempt to use it as part of a BACKGROUND job.

Entering PAUSE without a message causes the console to display a blank line before waiting for the carriage return.

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



INPUT PARAMETERS

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

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

Value	Access
D	Delete
L or R	List (for directories) and Read (for data files)
A	Add entry (for directories) and Append (for data files)
C or U	Change (for directories) and Update (for data files)
N	Cancels all access not explicitly granted (used without an accompanying value)

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

value

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

Value	Meaning
0	Cancel the access right
1	Grant the access right
	The default value is 1. That is, specifying an access character without a value grants the corresponding access.

user-list

User IDs for whom the previously-specified access rights apply. User IDs must consist of decimal or hexadecimal characters. Two special values are also acceptable for this parameter. They are:

WORLD	Special user ID (65535) giving all users access to the file.
*	Designator indicating that the access rights apply to all users currently in the file's access list.

The operating system limits each file to three user IDs in the access list. If you omit this parameter, PERMIT takes on the user ID associated with your interactive job.

DATA

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

DIRECTORY

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

MAP — Specifies that access information also applies to the map and volume label files in the pathname list. If you use the MAP parameter, you must specify the full pathname of any map files or volume label files in the pathname list. For example:

```
PERMIT :f0:R?* DLAU MAP
```

changes the access rights for all map files and volume label files on the volume (with the exception of R?SAVE which is unaffected by the MAP parameter). Notice that in this instance the Human Interface does not interpret the "?" as a wild card character.

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

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

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

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

DESCRIPTION

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

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

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

- Add or subtract users from a file's list of accessors. This list determines which users have access to the file.

- Set the type of access (access rights) granted to the users in the accessor list.

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

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

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

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

is the same as the combination:

In the first combination, the D1 overrides the D0.

You can use the N character to cancel all access to the file. If specified alone, it removes all user IDs from the accessor list. However, the N character can also be useful when changing access rights, if you don't remember the specified user's current access rights. In this case you can specify the N character first, to clear all the access rights, and follow it with other characters to grant the desired access. For example, if you want to grant list access only, you could specify "NL" instead of "D0A0C0L".

File access rights for remote files are computed somewhat differently than for local files. For information on remote files, see *iRMX Networking Software User's Guide*.

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

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

where

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

ERROR MESSAGES

- `<pathname>, accessor limit reached`

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

- `<pathname>, directory CHANGE access required`

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

- `<user ID>, duplicate USER control`

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

- `<character>, invalid access switch`

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

- `<invalid id>, invalid user id`

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

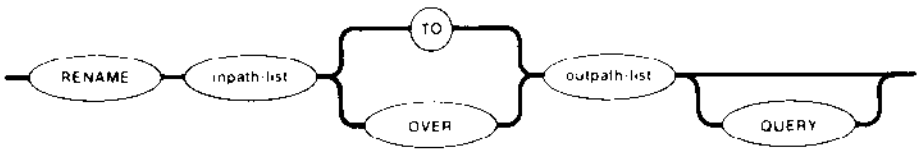
- `missing access switches`

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

- `no files found`

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

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:



x 321

INPUT PARAMETERS

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

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

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

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

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

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

OUTPUT PARAMETERS

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

```
<pathname>, already exists, DELETE?
```

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

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

outpath-list — List of new pathnames. Multiple pathnames must be separated by commas. Separating blanks are optional.

DESCRIPTION

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

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

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

Although RENAME can be used to rename an existing directory pathname TO a new pathname, it cannot be used to rename an existing directory OVER another existing directory (an E$DIR_NOT_EMPTY exception code is returned). For example:

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

NOTE

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

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

```
<old pathname>, renamed TO <new pathname>
```

or

```
<old pathname>, renamed OVER <new pathname>
```

ERROR MESSAGES

- `<old pathname>, DELETE access required`

  You cannot rename a file unless you have delete access to that file.

- `<new pathname>, directory ADD ENTRY access required`

  You cannot rename a file unless you have add-entry access to the destination directory.

- `<new pathname>, new pathname same as old pathname`

  You specified the same name for the input pathname as you did for the output pathname.

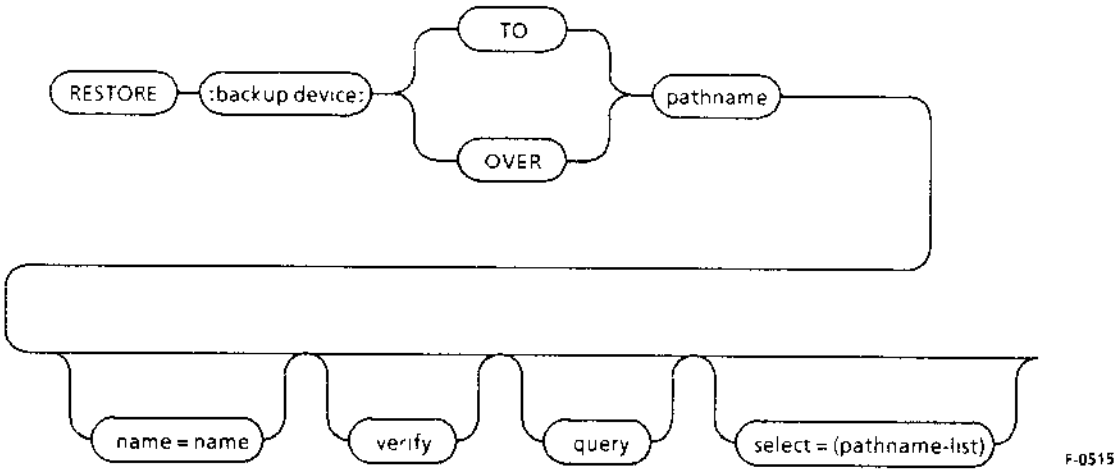
- `TO or OVER preposition expected`

  Either you used the AFTER preposition with the RENAME command or the number of files in your inpath-list did not match the number in your outpath-list.

This command transfers files from a backup volume to a named volume.

CAUTION

While you use this command, no other activity should be occurring on the volume to which you are restoring. If other users are accessing the volume during a RESTORE operation, the volume's data could become corrupted, possibly requiring the volume to be reformatted.



INPUT PARAMETERS

:backup device:

Logical name of the backup device from which RESTORE retrieves 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:

Entry	Action
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.

VERIFY

Verifies that BACKUP has produced a restorable set of volumes. Usually, when you use the VERIFY parameter you should specify the Byte Bucket (:BB:) as the output pathname. When you select VERIFY, no file is actually restored from the backup volume. Only the data on the volume is validated. RESTORE produces one of the following messages:

```
<pathname>, Verified
```

or

```
<pathname>, Directory Verified
```

NAME=name

NAME begins restoration from a specific data set. If no name is given, RESTORE restores only the first logical volume it encounters.

SELECT= (pathname-list)

A list of pathnames, separated by commas, designating the specific files or directories to be restored. The complete list must be enclosed in parentheses. The pathnames cannot include the logical volume name and must be the exact pathnames used in the BACKUP command. If you don't know the pathnames, use RESTORE with the VERIFY parameter and note the pathnames that appear on the screen. Then use RESTORE again, this time with the SELECT parameter and the pathnames you noted.

OUTPUT PARAMETERS

TO

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

```
<pathname>, already exists, OVERWRITE?
```

Enter one of the following in response to the query:

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

OVER

Restores the files from the backup volume over (replaces) the files on the named volume. If a file does not exist on the named volume, RESTORE creates a new file on the named volume. When you specify the OVER preposition, RESTORE does not prompt you for permission to overwrite existing files.

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. RESTORE allows the transfer operation to begin at any named data set or at any physical volume in a backup volume set. By using the SELECT parameter you can specify which files or directories will be restored.

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

When copying files, RESTORE reconstructs the following information:

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

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

The output volume which receives the restored files must be a named volume. You must have sufficient access rights to the files in that volume to allow RESTORE to perform all necessary operations. For RESTORE to create new files on a named volume, you must have add entry access to directories on that volume. For RESTORE to restore files over existing files, you must have add entry and change entry access to the files 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 II Restore Utility Vx.y
Copyright <year> Intel Corporation
```

where Vx.y is the version number of the utility. Then the command 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 next backup volume in the backup device.

Enter one of the following:

Entry	Action
Y, y, R, or r	Continue the restore process.
E or e	Exit from the RESTORE command.
N or n	Reprompt for a new volume.
Any other character	Invalid entry; reprompt for entry.

If you supply the requested volume, RESTORE starts restoring files from that volume and, if necessary, requests additional backup volumes. Once you supply the first backup volume, you must supply all the other backup volumes in the data set, in numerical order, when RESTORE requests them.

However, when RESTORE requests the first backup volume, you can supply a higher-numbered backup volume, if you know that all the files you want to restore reside on higher-numbered volumes. RESTORE will start restoring from that higher-numbered volume and maintain the proper directory structure for the files it restores. However, once you supply the first volume, you must still supply all the remaining backup volumes, in numerical order, when RESTORE requests them.

If a data file with the same pathname already exists when you use the TO proposition, RESTORE displays the following message:

```
<pathname>, Already Exists, OVERWRITE?
```

To continue or exit from the RESTORE command, enter one of the response characters listed above.

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

```
<pathname>, Restored/Verified
```

or

```
<pathname>, Directory Restored/Verified
```

If a "not restored" prompt is displayed, then a more detailed error message is printed.

ERROR MESSAGES

- `<pathname>, access to directory or file denied`

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

- `<backup device>, Backup Volume #<nn>, <date>, Mounted`
  `<backup device>, Backup Volume #<nn>, <date>, Required`

  `<backup device>, Mount Backup Volume #<nn>, Enter Y to Continue:`

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

- `<backup device>, Cannot Attach Volume`
  `<backup device>, <exception value> : <exception mnemonic>`

  `<backup device>, Mount Backup Volume #<nn>, Enter Y to Continue:`

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

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

  The BACKUP utility 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 II exception code encountered.

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

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

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

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

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

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

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

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

- `<backup device>, invalid backup device`

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

- `<backup device>, Not a Backup Volume`

  `<backup device>, Mount Backup Volume #<nn>, Enter Y to Continue:`

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

- `<pathname>, Not Restored`

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

- `output specification missing`

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

- `<pathname>, READ access required`

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

- `<pathname>, too many input pathnames`

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

- `Select List Too Long`

  The pathname list you supplied with the SELECT parameter exceeded 255 bytes. In this case, you should invoke RESTORE again with a shorter pathname list.

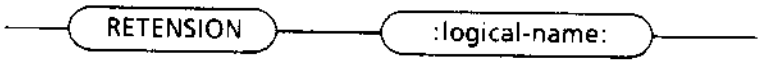
- `Invalid Select : select = (filename [, filelist])`

  You supplied a single pathname with the SELECT parameter and it was not enclosed in parentheses.

- `select, unrecognized control`

  You supplied a list of pathnames with the SELECT parameter and the list was not enclosed in parentheses.

This command retensions a tape. Occasionally, while a tape is being read or written, it winds or lays unevenly on the spool. This command causes a tape to be wound evenly on the spool. The format of the command is



F-0831

INPUT PARAMETERS

:logical-name:	A logical name of a device that supports the RETENSION command. Currently, the only devices that support the RETENSION command are tape devices.

DESCRIPTION

The RETENSION command evenly winds tape media. Invoking this command causes a fast-forward operation to occur on the tape associated with :logical-name: until the end of the tape is reached. After the tape fast-forwards, a rewind operation occurs back to the load point. These two operations together provide for an evenly wound tape.

EXAMPLE

The following example invokes the RETENSION command on a physical tape device whose logical name is :tape:.

```
-
```

The command displays the following message:

```
Starting RETENSION Operation
```

ERROR MESSAGES

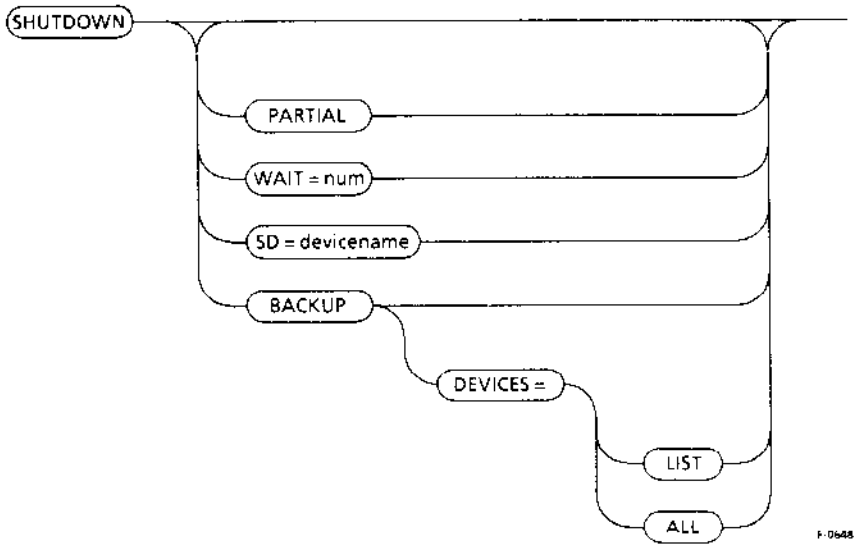
- `Invalid logical name`

  The logical name does not exist.

- `Device does not support retension`

  This error message is caused by the parameter :logical-name: pointing to a file (E$IFDR) or a device other than a tape drive (E$DDR).

This command provides for the orderly shutdown of the system. All Human Interface users are warned of an impending shutdown, at fixed intervals until the shutdown takes place. The format of this command is



INPUT PARAMETERS

PARTIAL — Requests a partial shutdown. This parameter only locks HI terminals and aborts all HI jobs other than the operator's.

WAIT = nn — Sets the delay period requested before shutdown procedures begin. The wait value is entered in minutes. The maximum value which may be entered is 30 and the default value is 10. A value of zero indicates no delay.

SD = device name — Defines the system device containing the system directory and the volume master files. The device name must be a named volume that is currently attached. The default value is :SD:. Even though the system device can be different than the default, do not change it unless you have a specific reason to do so.

BACKUP — Instructs the SHUTDOWN utility to create a backup of the system device volume master files and any other devices specified in the DEVICES parameter.

DEVICES = — Marks the designated devices as shutdown. If the BACKUP parameter has been entered, the fnode file on all the specified devices is backed up. The devices to be shutdown can be specified in two ways:

- list — A list of devices to be marked as shutdown.
- ALL — All attached EIOS logical named devices are marked as shutdown.

DESCRIPTION

The SHUTDOWN command provides the system with an orderly shutdown procedure. All Human Interface terminals are locked, and the associated Human Interface User Job Tree is deleted. This command can only be invoked by the system manager (user 0).

The SHUTDOWN utility provides a number of options such as a time delay or only a partial shutdown. When you are entering these parameters it is not necessary to enter the entire parameter. It is only necessary to enter enough to create a unique selection. For example, you might enter:

to indicate a 10 minute wait, a backup of the fnode file, and the marking of :dev1: and :dev2: as shutdown on the volumes.

When SHUTDOWN is invoked all named devices, including the system device, that have been logically attached using the Extended I/O System are detached. This closes all file connections on the devices, and flushes all EIOS and BIOS buffers associated with these devices. These volumes are marked as properly shutdown and the following message is displayed:

```
:SD:, outstanding connections to device have been deleted
***SHUTDOWN COMPLETED
```

The system manager job tree is then deleted.

If you enter SHUTDOWN without parameters, SHUTDOWN displays:

```
***SYSTEM WILL BE SHUT DOWN IN 10 MINUTES(S)
```

Any errors detected during the shutdown process cause the utility to abort and display the message:

```
***  SHUTDOWN ABORTED
```

If a syntax error is encountered in the invocation of SHUTDOWN, the proper usage is displayed as follows:

```
USAGE:  SHUTDOWN [WAIT=nn] [SD=sys_dev] [BACKUP] [DEVICES=list] [PARTIAL]
```

The utility then aborts and returns control to the system command level. (A complete list of syntax errors is given in the "Error Messages" section.)

If SHUTDOWN is unable to delete one of the Human Interface users, it displays the following message:

```
****    unable to delete user, <hi_user>
****    Continue? (Y/N)
```

where <hi_user> is the user name as defined by the Human Interface. This message is displayed after waiting five minutes for the user job to be deleted.

During the shutdown process, SHUTDOWN catalogs the R?SHUTDOWN object in the root directory to ensure that first-level jobs will be able to close down and exit in an orderly fashion. When SHUTDOWN is aborted the R?SHUTDOWN object is uncataloged.

Wait

You may specify a delay in the invocation to allow time to complete any cleanup procedures. When SHUTDOWN is invoked a warning message is issued at five minute intervals until less than five minutes remain, and from then on at one minute intervals. The message displayed is

```
*** SYSTEM WILL BE SHUTDOWN IN nn MINUTE(S)
```

where nn is the time remaining before shutdown.

Partial Shutdown

The system manager may need to delete only a limited number of Human Interface users such as when backing up a disk. This is possible by including PARTIAL in the invocation. When the system is ready to return to general use, the UNLOCK command (described later in this chapter) can be entered to reinitiate all users.

Backup

The system manager can request that the system volume fnode file be copied to its duplicate file: R?SAVE, by invoking the BACKUP parameter. If the DEVICES parameter is entered, the fnode files of the logical device names specified are also backed up. When a successful backup has been made, the Human Interface displays:

```
****BACKUP OF VOLUME FILES ON log_dev COMPLETED
****BACKUP OF VOLUME FILES ON sys_dev COMPLETED
```

where log_dev is the logical name of the device(s) specified in the DEVICE option, and sys_dev is the logical name of the system device as specified in the command invocation. If the DEVICES option is not selected, only the system device is backed up. :SD: is the default system device.

Any errors detected while trying to backup the files are displayed immediately as follows:

```
***  error in device fnode <number>
***  SHUTDOWN COMPLETED
```

The system directory on the system device volume is stamped to enable the Human Interface initialization to set the system clock the next time the system is booted. This ensures that the system clock, used in time stamping files, always moves forward chronologically.

Devices

The system manager can indicate that specified logical named devices be marked as shutdown by including the DEVICES parameter in the invocation. If no devices are specified, all EIOS logical named and remote devices are shutdown. Whenever the DEVICES parameter is entered, the BACKUP option must also be entered. This causes the fnode files on the designated devices to be backed up. If an error is detected while detaching a named device, the following message is displayed:

```
*** error detaching device, <logical dev name>
```

An error encountered when marking a volume as shutdown, causes this message:

```
*** error marking shutdown device, <dev>
```

When an error is detected during the backup and marking process of a logical named device, as opposed to the system device, the processing continues.

Aborting the Shutdown

During the operation of the SHUTDOWN Utility, the system manager can enter CONTROL-C to abort the procedure. SHUTDOWN can be aborted only at the completion of a logical operation. That is, SHUTDOWN can only be aborted after all the terminals have been locked, but not during the terminal lock process. If you use CONTROL-C to abort SHUTDOWN, the UNLOCK command must be used to free each terminal. The following logical operations are defined in the SHUTDOWN Utility.

Operation	Function
Terminal locking	Locks all HI terminals.
Warnings	Issues a warning every 5 minutes until 5 minutes before shutdown, then issues a warning every minute.
Job deletion	Deletes all HI user jobs, excluding the caller's job.
Time stamping	Time stamps the system directory.
Backup	Backs up all fnode files.
Detaching	Detaches all EIOS named and remote devices.
Marking	Marks the volume as shutdown.
Delete job tree	Deletes the caller's job tree.

ERROR MESSAGES

The errors listed in this section are syntax errors not previously explained in the "Description" section.

- `<keyword>, unknown keyword or switch`

  A keyword other than PARTIAL, WAIT, SD, BACKUP or DEVICES was encountered.

- `illegal keyword`

  A switch was used as a keyword.

- `illegal value`

  A keyword was assigned an illegal value.

- `<system_dev>, not a logical device name`

  The name you entered is not cataloged as a logical device.

- `<sys_dev>, not a named device`

  The logical device name entered is not a named device.

- `<log_dev>, not a logical device name`

  The name you entered is not cataloged as a logical device.

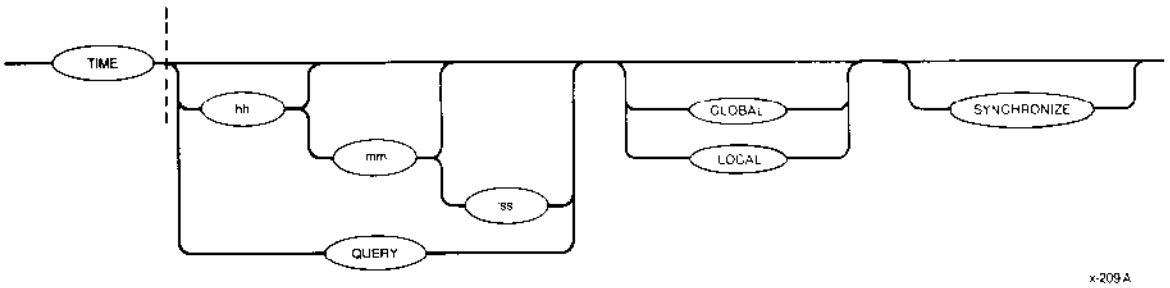
- `<log_dev>, not a named device`

  The logical device name entered is not a named device.

This command is for users with a customized CLI. It reads and executes a set of commands from a file in secondary storage instead of from the console keyboard. If you are using the Intel-supplied CLI, SUBMIT is a CLI command supporting all the CLI features. The syntax and rules for SUBMIT as a CLI command or an HI command are almost the same. The only exception is that when using SUBMIT as an HI command, none of the CLI features, such as aliasing, can be included in the SUBMIT file. For a complete description of SUBMIT, see Chapter 3.

This command allows users with a customized CLI to change their user IDs to the system manager ID. If you are using the Human Interface CLI, this command is available as a CLI command. For a complete explanation of SUPER, see Chapter 3.

This command sets the local or global system time or displays the current time. The format is as follows:



INPUT PARAMETERS

hh — Hours specified as 0 through 23.

mm — Minutes specified as 0 through 59. If you omit this parameter, 0 is assumed.

ss — Seconds specified as 0 through 59. If you omit this parameter, 0 is assumed.

QUERY — Causes TIME to display the current date, time and clock type followed by the prompt:

```
TIME:
```

TIME continues to issue this message until you enter a valid time or the letter E (or e) to exit.

GLOBAL — Applies only to systems with hardware clock/calendar components. Such clock/calendar components are usually powered by batteries so they continue keeping time when power to the system is turned off. These clock/calendar components are referred to as global clocks. This parameter causes TIME to display or set the time portion of the global time-of-day clock. Any user can display the current value of the global clock, but only the system manager can set the global clock. If the global clock is modified, the local clock automatically takes on the new value of the global clock. LOCAL is the default if the LOCAL and GLOBAL parameters are omitted.

TIME displays an error message if you specify this parameter and your system does not have a global clock/calendar.

LOCAL	Causes TIME to access (to display or set) the time portion of the local time-of-day clock maintained by the operating system. All users may display and set the local clock to a new value. Local is assumed if the LOCAL and GLOBAL parameters are omitted.
SYNCHRONIZE	Applies only to systems with global clock/calendars. This parameter causes TIME to set the time portion of the local time-of-day clock to the current time value in the global clock. If you are modifying the global clock, this parameter is unnecessary. TIME displays an error message if you specify this parameter and your system does not have a global clock/calendar.

DESCRIPTION

You must separate the individual time parameters with colons.

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

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

where dd mmm yy indicates the date, hh:mm:ss indicates the time, and clock type designates either a global or local clock type.

If you have a system without a global clock/calendar, whenever you start up or reset the operating system, the time is automatically set to the time you last accessed the :SYSTEM: directory plus the time that elapsed since the system was started. You can reset the time to any acceptable value.

If your system has a global clock/calendar and the operating system is configured to recognize it, the local clock is automatically set to the time maintained in the global clock when you turn on or reset your system.

The TIME command enables you to set and/or display the time portion of two time-of-day clocks: the local clock and the global system clock. You access the local clock by specifying the LOCAL parameter; you access the global clock by specifying the GLOBAL parameter. If neither LOCAL nor GLOBAL is specified, the local clock is accessed by default. Any user can display the time and date portion of the local and global clocks.

However, only the system manager can set the time portion of the global clock. If the system manager sets the global clock to a new value, the local clock will automatically be set to that value.

ERROR MESSAGES

- `<time>, invalid time`

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

- `<parameter>, invalid syntax`

  You specified an illegal combination of parameters such as, both a time and the QUERY parameter.

- `only the system manager may set the global clock`

  You specified the GLOBAL parameter, but you are not the system manager.

- `E$SHARE, global clock busy`

  You attempted to access the global time-of-day clock while another job was accessing it. Try the command again.

- `<exception value>:<exception mnemonic>, while getting system time`

  The indicated exception occurred while the TIME command was getting the time from the global time-of-day clock.

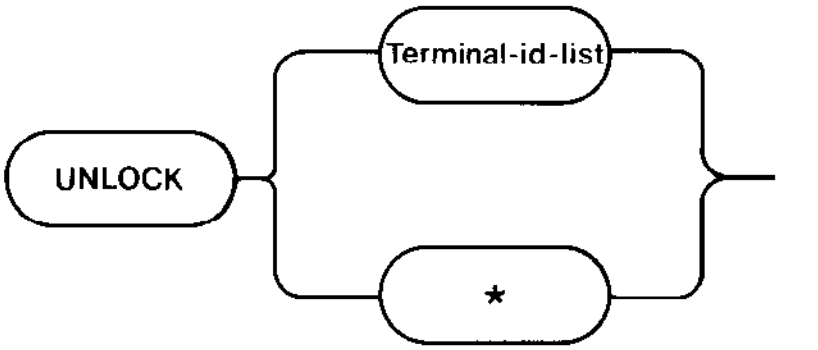
- `E$INVALID$DATE, global date read was invalid`

  The date returned from the global clock was invalid. This condition will usually occur when the global clock has never been initialized or when power to the clock has been interrupted. The BIOS system call GET$GLOBAL$TIME gets the date from the global clock, which the TIME command then displays.

- `E$INVALID$TIME, global time read was invalid`

  The time returned from the global clock was invalid. This condition will usually occur when the global clock has never been initialized or when power to the clock has been interrupted. The BIOS system call GET$GLOBAL$TIME gets the time from the global system clock, which the TIME command then displays.

This command enables users who have been locked out of the system to log back on. The format of the UNLOCK command is



INPUT PARAMETERS

Terminal-id-list	A list of the terminals to be unlocked.
*	A wildcard indicating that all configured terminals are to be unlocked.

DESCRIPTION

The UNLOCK command can be used only by the system manager to unlock terminals that were locked out by either the LOCK or SHUTDOWN commands. This command allows all the terminals or only those listed in the invocation to log back into the system. UNLOCK causes the Human Interface to initiate the log in procedure for the appropriate terminal, if logged off.

When the terminal is unlocked, the following message is displayed:

```
unlocked
<terminal-id>, unlocked
```

If the terminal specified is not configured into the system, the UNLOCK command issues this message:

```
<terminal-id>, not found
```

ERROR MESSAGES

- `not multi-user system`

  You entered more than one terminal number in a system that is not a multi-access system.

- `unlock not allowed to non-SUPER users`

  You are not the system manager and therefore, are not entitled to issue this command.

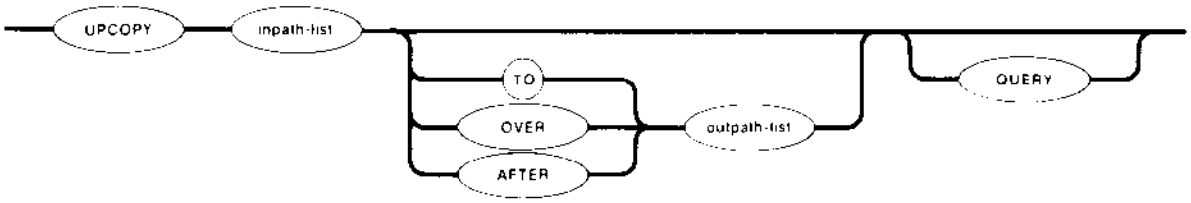
- `parameters required`

  You entered UNLOCK with no parameters.

- `<terminal-id>, not found`

  The terminal-id you specified is not configured into the system.

This command copies files from a Series II, II,or IV Microcomputer Development System to iRMX II secondary storage using the iSDM monitor.



x 323

INPUT PARAMETERS

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

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

```
<in-pathname>, copy up TO <out-pathname>?
<in-pathname>, copy up OVER <out-pathname>?
<in-pathname>, copy up AFTER <out-pathname>?
```

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

Entry	Action
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 the sequence.

OUTPUT PARAMETERS

TO — Copies the development system file or files to a new iRMX II file or files in the listed sequence. If the output file already exists, UPCOPY displays the following message:

```
<pathname>, already exists, OVERWRITE?
```

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

If no preposition is specified, TO :CO: is assumed. If more input files than output files are specified in the command line, the remaining input files will be appended to the end of the last listed output file.

OVER — Copies the listed development system input file or files over existing iRMX II 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 development system input file or files after the end-of-data on an existing iRMX II output file or files in the listed sequence.

outpath-list — One or more pathnames of the iRMX II 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 II console screen.

DESCRIPTION

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

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

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

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

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

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

When the UPCOPY command is executing, the monitor disables interrupts. This action affects services such as the time-of-day clock. Also, the operating system is unable to receive any characters that you type-ahead while the UPCOPY command is executing.

ERROR MESSAGES

- `<pathname>, ISIS ERROR: <nnn>`

  A development system error occurred when UPCOPY tried to transfer the file from the Microcomputer Development System. Refer to the *Intellec Series IV Operating and Programming Guide* for a description of the resulting error code.

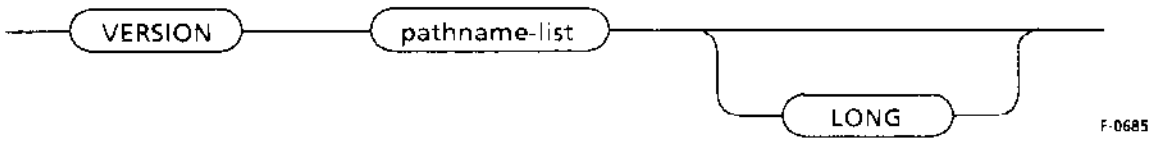
- `ISIS link not present`

  The iRMX II system is not connected to the development system via the monitor.

- `<pathname>, UPDATE or add access required`

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

This command displays the version number of a file if that file has a version number. The file can be an object file or library. The format of this command is as follows:



INPUT PARAMETERS

pathname-list	One or more pathnames, separated by commas, or a wild-card expression of commands for which a version number is desired.
LONG	Displays all the version numbers residing in the input file.

DESCRIPTION

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

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

where

<pathname>	Pathname of the file containing the command.
<module-name>	Name of the specified command or library; Intel-supplied commands have names as listed in this manual.
x.y	Version number of the command.

You can use VERSION to determine the version number of any Human Interface command. You can also use it to determine the version numbers of commands that you write. If the file is a library, the command shows the current and previous version numbers. However, for VERSION to work on your commands, you must include a literal string in the command's source code to specify the name of the command and its version.

The string must contain the following information:

```
'program_version number=xxxx',
'program_name=yyyy...yyy',0
```

where

program_version_number=	You must specify this portion exactly as shown (lower case, underscore separating the words, no spaces).
xxxx	Version number of the product. This can be any four characters, but it must be exactly four characters long.
program_name=	This portion is optional. However, if you want VERSION to recognize and display the program name, you must specify this portion exactly as shown.
yyyy...yyy	Name of the command. This name can be any number of characters.
0	The literal string must be terminated with a byte of binary zero.

An example of such a literal string is:

```
DECLARE version (*) BYTE DATA('program_version_number=V8.5',
                              'program_name=MYPROG',0);
```

If your program includes this declaration, when you invoke VERSION, it will display the following information:

```
<pathname>, MYPROG version is V8.5
```

A literal string that does not include the program name is:

```
DECLARE vers2(*)        BYTE DATA('program_version_number=1986',0);
```

If your program includes this declaration, when you invoke VERSION, it will display the following information:

```
<pathname>, version is 1986
```

ERROR MESSAGES

- `<pathname>, does not contain a program version number.`

  The command you specified does not contain version number information.

- `<pathname>, is not an object module.`

  The pathname you specified does not represent a file containing executable object code.

This command lists the current user's identification and access rights.

x-666

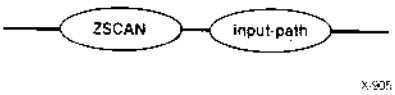
DESCRIPTION

This example shows the output from WHOAMI:

```
-WHOAMI <CR>
User id: # 5
Access id's:  5, WORLD
```

The number after User id is the user's ID number. The numbers after Access id's are the IDs of other users who have granted the user access to their files.

This command reads an object file or an object library and displays the identification number of all iRMX II ZAPs (fixes) that have been applied to that file.



X-905

INPUT PARAMETER

input-path — The pathname of the object or object library to be scanned. The pathname cannot contain wildcard characters. The pathname must specify a file, not a directory.

DESCRIPTION

Fixes for problems discovered in the operating system software are distributed through the iRMX II Update Service. Intel refers to these fixes as "ZAPs". ZAPs are new modules that replace the corresponding modules in the operating system.

Each update diskette contains an accumulation of all ZAPs issued during the current release of the operating system. When you install the latest update, all ZAPs (from the current update and from previous updates) are automatically applied to your system.

The ZSCAN command allows you to check which ZAPs have been applied to an object file or an object library. All ZAPs are marked by a unique identifier string. ZSCAN finds occurrences of these strings and returns information about the associated ZAPs.

When you invoke ZSCAN, you must specify an object file or an object library. You cannot invoke the command to find all of the ZAPs applied within a specified directory. Furthermore, you cannot use wildcard characters in the pathname of the file to be scanned.

By default, the iRMX II system object files are not accessible to user WORLD. Therefore, if you intend to use ZSCAN on a bootable system object file, you must grant user WORLD read access rights to that file (using the Human Interface PERMIT command) or invoke ZSCAN from the SUPER mode.

OUTPUT DISPLAY

Upon successful execution, the ZSCAN command displays one of the two following messages.

When ZSCAN encounters ZAPs:

```
<filename>, has the following ZAP(s) applied:
<zap id>, <class>: for <product>, <layer>, <release>
                    .
                    .
                    .
<zap id>, <class>: for <product>, <layer>, <release>
```

where

<filename>	the name of the file being scanned
<zap id>	the identification code for the ZAP: ZPCxxx where: Z = a ZAP identifier string P = product code B = iRMX II MB II R = iRMX I X = iRMX II N = iRMX I-based iRMX-NET P = iRMX II-based iRMX-NET C = the class of ZAP A = fully evaluated B = developer tested xxx = the unique number assigned to the ZAP
<class>	the class of the ZAP. Class A indicates a supported ZAP distributed through the iRMX II Update Service. Class B indicates an un-supported ZAP with limited distribution.
<product>	the product the ZAP is associated with.
<layer>	the layer of the operating system (e.g. Nucleus, BIOS, etc.) that the ZAP pertains to.
<release>	the release level of the operating system layer that the ZAP pertains to.

When ZSCAN encounters no ZAPs:

```
<filename>,          No ZAPs applied.
```

where

`<filename>` the name of the file being scanned.

ERROR MESSAGES

- `USAGE: zscan <object file>`

  No filename was specified when the command was entered.

- `<filename>, file does not exist`

  There is no file with the pathname specified in the command.

- `<filename> is not an object module`

  The file specified in the command is not an object module and cannot be scanned for ZAPs.

- `<parameter>, Unrecognized control`

  An unknown parameter was specified when the command was entered.

CHAPTER 5
HUMAN INTERFACE EXAMPLES

5.1 OVERVIEW

This chapter shows examples of some of the most frequently used Human Interface commands. It is written to introduce first time operators to the basic techniques needed to use the Human Interface commands. If you are a more experienced operator, you may want to skip this chapter.

5.2 COMMAND EXAMPLES FORMAT

To make it easier to follow the interactive dialogue between the operator and the Human Interface in the examples, the user keyboard entries are printed in or bolded if within a gray box. All other items displayed in the examples are Human Interface command output. For instance, in the following example,

```
-copy samp to test <CR>
samp copied TO test
-copy test <CR>
aaaaa
bbbbb
test copied TO :CO:
-
```

the bolded items are operator command entries; all other characters and lines are output by the Human Interface or the supplied commands.

Control characters, such as (CONTROL-Z), are enclosed in parentheses in the examples to indicate that such entries are not echoed at the console screen as they are entered. Do not actually enclose control key entries in parentheses.

5.3 BEGINNING A CONSOLE SESSION

You can begin an interactive dialogue with the Human Interface after the initial program displays a sign-on message at your console screen. Although the sign-on message is a system configuration option, the message supplied with the default initial program of the Human Interface is as follows:

```
iRMX II HI CLI Vx.y: USER = <userid>
Copyright <years> Intel Corporation
-
```

This message tells you the Human Interface is running; it also tells you your user ID. The hyphen (-) is a CLI default prompt indicating that the initial program is ready to accept your first command line. Begin entering a command on the same line as the prompt. For example:

```
-
```

5.4 CREATING A SIMPLE DATA FILE

You can use the COPY command to create data files during a console session. Assume you wish to create a file called ALPHA and write two lines of data into the file. Also assume you wish the data file to be listed under your default directory, :$:. Enter the following command and data:

```
-
```

The HI responds:

```
: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 bbbbb) to a new file named ALPHA. Because you did not preface the file name with a directory name, COPY places the file ALPHA in your default directory.

The command does not prompt you for the data lines; you simply begin entering data after you press RETURN at the end of the command line. Your (CONTROL-Z) entry writes an end-of-file mark at the end of your data to inform the COPY command that there is no more data to be copied.

Note that after you enter the last line of data, you must press the RETURN key <CR> before you enter a (CONTROL-Z) to insert an end-of-file. Otherwise, the (CONTROL-Z) will be ignored.

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
ccccc
ddddd
:ci: copied TO alpha
-
```

Now, assume that when you entered the COPY command line, the Human Interface sent you the following message and query:

```
-copy :ci: to alpha <CR>
alpha, already exists, OVERWRITE?
```

Whenever you create a new data file, the COPY command expects a new pathname rather than one already listed in the directory file. If your entry to the query is:

```
alpha, already exists, OVERWRITE? y <CR>
```

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

If your response to the query is:

```
alpha, already exists, OVERWRITE? n <CR>
-
```

your COPY command is ignored and the Human Interface prompts for a new command.

5.5 COPYING FILES

The COPY command options provide a number of different ways for you to copy existing files. You exercise these options either by specifying one of the TO/OVER/AFTER prepositions, by the way in which you specify your input file and output file pathname lists, or by a combination of both techniques. The services of the COPY command include:

- Copying files on a one-for-one basis.
- Displaying the contents of files at the console screen.
- Creating multiple copies of the same file.
- Copying data from multiple files to a new or existing file.
- Replacing data in one file with data from another file.
- Adding data from one or more files to the end of the data in another file.
- Combining one-for-one file copying with file concatenation in a single COPY command.

The examples that follow show you how to use these services. They also call your attention to certain file handling considerations when using the COPY command.

5.5.1 Copying 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 wish to copy files A1, A2, and A3 to files B1, B2, and B3 respectively. Enter the following command:

```
-
```

The Human Interface responds:

```
a1 copied TO b1
a2 copied TO b2
a3 copied TO b3
-
```

You can also make use of the wild card feature when copying files. If the files A1, A2, and A3 are the only files in the directory that begin with the character "A", you can use the following command to perform the same operation:

```
-copy a* to b* <CR>
a1 copied TO b1
a2 copied TO b2
a3 copied TO b3
-
```

The asterisks in the command are the wild card characters. In this instance, the command copies all files in the default directory that start with the character "A" to new files starting with the character "B". If files other than A1, A2, and A3 also begin with the character "A", this command will copy them also.

When you copy files, you can specify wild card characters (as in the previous example), lists of file names (as in the example before that), or a combination of both. However, some of the possible combinations are invalid. When copying files, remember the following rules:

- If you specify multiple input pathnames and a single output pathname, file concatenation takes place. If the output parameter is simply a directory with no wild card in its pathname, then the Human Interface copies all the files listed in the input parameter into the directory. Each file keeps its original name in the new directory (such as alpha).
- If you specify multiple output pathnames, you must specify the same number of input pathnames as output pathnames. Specifying more input pathnames than output pathnames results in an error message. For example, the command:

  ```
  -                                    (invalid)
  ```

  returns an error message. The command:

  ```
  -                                    (invalid)
  ```

  also returns an error message. Refer to the "Inpath-List and Outpath-List" section of Chapter 2 for more information.

5.5.2 Displaying the Contents of Files

When you perform a number of file manipulations during a single session, it is occasionally advisable to display a file's contents at the terminal before proceeding further. Assume you wish to display the contents of a file named ALPHA that is contained in your default directory. Simply enter the command:

```
-copy alpha <CR>
aaaaa
aaaaa
alpha copied TO :co:
-
```

This COPY command example uses the default preposition (TO) and default output file (:CO:), which means that the command copies the output to the console screen.

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

CONTROL-W	Puts the terminal into scrolling mode. In this mode, output stops after a single screen of data appears. Entering another CONTROL-W displays the next screen of data.
CONTROL-S	Stops the data from scrolling off the screen until you press a CONTROL-Q.
CONTROL-Q	Resumes scrolling of listed data until the end-of-file is reached or you enter a CONTROL-C.
CONTROL-C	Cancels listing of the data and returns control to the Human Interface, which prompts for a new command.

5.5.3 Replacing Existing Files

There may be occasions when you wish to update the contents of an existing file. One way to do this updating is to create a new file and then replace the contents of the old file with the new data. Although you can use the RENAME command to perform this operation, this section shows how to replace the contents of a file with the COPY command's OVER preposition.

Assume the following conditions:

- You have a file named ALPHA that is accessed under that name by a number of different programs. ALPHA has outmoded data.
- Since you cannot change the name without also modifying the programs that access ALPHA, you must retain the name but update the outmoded file contents.

Enter the following command sequence:

```
-copy :ci: to temp <CR>
nu nu nu nu <CR>
nu nu nu nu <CR>
(CONTROL-Z)
:ci: copied TO temp
-copy temp over alpha <CR>
temp copied OVER alpha
-copy alpha <CR>
nu nu nu nu
nu nu nu nu
alpha copied TO :co:
-
```

The last COPY ALPHA command lists the file at the terminal to show that the old file contents have been successfully replaced.

You could have used the TO preposition in the COPY command to write TEMP over ALPHA; but since the Human Interface always expects a new output file when the TO preposition is used, this would have caused unnecessary keystrokes, as shown in the following:

```
-copy temp to alpha <CR>
alpha, already exists, OVERWRITE? y <CR>
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 <CR>
newnewnew <CR>
(CONTROL-Z)
:ci: copied OVER alpha
-
```

5.5.4 Concatenating Files

Concatenation is the process of combining a number of files by appending them in sequence into a single file. You can use the COPY command in several ways to concatenate files:

- by specifying the AFTER preposition in the command line
- by specifying multiple input pathnames and a single output pathname (if the output pathname is a directory, concatenation does not occur)
- by using a combination of both techniques

Assume you have four existing files named A, B, C, and D respectively, and you 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 <CR>
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 <CR>
a copied TO all
b copied AFTER all
c copied AFTER all
d copied AFTER all
-
```

In this example, file A is copied to ALL and the remaining input files are automatically appended to the end of ALL.

You can save keystrokes when listing a series of files on the screen by using this automatic concatenation in a single command line. Assume you wish to list files named ALPHA, BETA, and GAMMA. Enter the following command, using the default TO preposition and default output file (:CO:):

```
-copy alpha,beta,gamma <CR>
aaaaa
aaaaa
alpha copied TO :co:
bbbbb
bbbbb
beta copied AFTER :co:
ggggg
ggggg
gamma copied AFTER :co:
-
```

When data sequence and/or data format are important in a concatenated file, remember that all copy operations are performed in the sequence you specify in the command line.

Assume you have formatted data in a group of files named A, B, C, D, and E, and you wish to concatenate their contents into a new file named SQUARE in that sequence. However, if you list the input files on the command line in a haphazard sequence, as follows:

```
-
```

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:

b,a,d,c,e	a,b,c,d,e
sequence	sequence
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.

5.6 DELETING 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 <CR>
alpha, deleted
beta, deleted
-
```

Now, assume that you entered the following command line and received the following error message:

```
-delete ay,bee,key <CR>
ay, deleted
bee, deleted
key, does not exist
-
```

The error message for the KEY file tells you one of three things:

- There is a spelling error in the name of the KEY file.
- The file does not exist.
- The file exists in a directory other than the one you are currently accessing (see the directory examples later in this chapter).

5.7 USING DIRECTORIES

A directory is a kind of file under which you assign and maintain other files or directories. It is distinguished from a data file by a directory heading that is automatically created when you create a new directory. Under that heading, the directory maintains a formatted list of the files and directories it contains. This heading is updated whenever you assign new files to the directory. Directories provide you with a convenient and efficient technique for organizing large numbers of files into logical groupings. Creating your own directories aids you in two ways:

- It allows you to organize your files into logical groupings. This capability eases the task of maintaining large numbers of files on the system.

- It reduces the possibility of accidental destruction of files, either by yourself or other system users.

A directory contains a list of all files assigned under its name, which you can display by using the DIR command. Optional DIR command parameters also allow you to access and display other pertinent information about each file, such as file size and other file attributes.

Previous command examples in this chapter, when creating and accessing files, have used the default directory configured for your user ID. The following examples show you how to create and use your own directories for easier file management.

5.7.1 Creating a New Directory

Whenever you wish to group a series of files under a single topical structure, you normally create a new directory in which to assign them before creating the files themselves. (You can also move existing files under a new directory name by using the RENAME or COPY commands.)

You create new directories by using the CREATEDIR command to specify a list of directory names for the new directories. You will find it easier to keep track of both your directories and files if you use directory names that give some hint of a directory's topical structure.

Assume you wish to create two directories named MYTEST and NUTEST under which you will assign several practice files. Enter the following command:

```
-createdir MYTEST,NUTEST <CR>
MYTEST, directory created
NUTEST, directory created
-
```

Once you create directories and data files, you can enter their pathnames in either lowercase or uppercase characters in subsequent commands; the Human Interface commands are not case sensitive.

5.7.2 Referring to a Directory

After you create a new directory, all named files or directories that you assign to that directory will have a hierarchical relationship to this "parent" directory. This relationship to the parent is called a path. When you wish to access any file or other directory assigned to the parent, you must specifically identify the path in the form of a pathname in your command.

For example, assume your default directory has a directory named NUTEST under which you have another directory named SAMP. SAMP, in turn, has a data file named TEST. NUTEST is then the parent directory for the SAMP directory and SAMP, in turn, is the parent for the TEST data file. In a command, the pathname for the SAMP directory would be NUTEST/SAMP, where the slash characters separate the individual hierarchical components of the pathname. The pathname for the TEST data file would be NUTEST/SAMP/TEST.

If the files are contained in your default directory, you can refer to them without specifying a logical name as a prefix. When you enter the pathname:

```
NUTEST/SAMP/TEST
```

the Human Interface automatically appends the prefix :$: to the beginning. However, if the files are contained in a directory other than your default directory, you must enter the complete pathname for the file. For example, if the files reside on a device whose logical name is :AD3:, you must include this logical name as the prefix portion of the pathname, as follows:

```
:AD3:NUTEST/SAMP/TEST
```

If you omit the :AD3: portion, the Human Interface assumes the files reside in the default directory.

5.7.3 Adding New Entries to a Directory

Previous data file examples in this chapter used the default directory (as configured for your system) for all file creation and access. Consequently, each example that created a new file or accessed an existing file specified only the last component of the file's pathname; it did not need to specify a logical name or intermediate pathname components. However, whenever you wish to create a new data file to be assigned to a specific directory, you must precede the filename with the directory name and separate the two names with a slash (/), as described in the previous subsection. You might also need to specify a logical name, if there is a logical name assigned to part of the pathname. Such a case is when you are copying across volume boundaries (discussed later in this chapter).

For example, assume you wish to create files named SAMP1 and SAMP2 and assign them to the MYTEST directory (MYTEST resides in your default directory). Enter the following commands:

```
-copy :ci: to mytest/samp1  <CR>
aaaaa  <CR>
(CONTROL-Z)
:ci: copied TO mytest/samp1
-copy :ci: to mytest/samp2 <CR>
bbbbb <CR>
(CONTROL-Z)
:ci: copied TO mytest/samp2
-
```

Remember that once you have added files to a specific directory, every subsequent operation involving those files must specify a preceding directory name and the slash separator (unless you change your default directory, as described in a later section). For example, assume you want to delete files SAMP1 and SAMP2 from the MYTEST directory. You might enter the following command:

```
-delete mytest/samp1,samp2  <CR>
mytest/samp1, deleted
samp2, file does not exist
-
```

The Human Interface issues the "file does not exist" message for SAMP2 because it looked for the file in your default directory instead of the MYTEST directory. The correct command line entry should have been:

```
-
```

so that the Human Interface would search the correct directory for each listed file.

5.7.4 Creating a Directory Within a Directory

In the same manner that you create new directories in your default directory, you can also create new directories in other directories, thereby expanding the file hierarchy. For example, assume you have data files ALPHA, BETA, and GAMMA assigned to the MYTEST directory and now wish to add a new directory file named URTEST to the directory. Enter a CREATEDIR command, as follows:

```
-createdir mytest/URTEST <CR>
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 <CR>
nononon <CR>
nononon <CR>
(CONTROL-Z)
:ci: copied TO mytest/urtest/nomor
-
```

The "MYTEST/URTEST" sequence is the path from your default directory to the URTEST directory, and the "MYTEST/URTEST/NOMOR" sequence is the path from your default directory to the NOMOR file. When you use file-handling commands, you must always specify a path to the file, either a path from your default directory to the file, or a path from some other known point (such as from the root directory for another device). For example, assume you have another data file in URTEST named SUMOR and wish to list both NOMOR and SUMOR on the console screen. Enter the following command and specify the pathname for each file:

```
-copy mytest/urtest/nomor,mytest/urtest/sumor <CR>
nononon
nononon
mytest/urtest/nomor copied TO :co:
sumsumsum
sumsumsum
mytest/urtest/sumor copied TO :co:
-
```

If the directory MYTEST resides on a device (for example, :F6:) other than your default device, you would specify the previous command as follows:

```
-copy :f6:mytest/urtest/nomor,:f6:mytest/urtest/sumor <CR>
nononon
nononon
:f6:mytest/urtest/nomor copied TO :co:
sumsumsum
sumsumsum
:f6:mytest/urtest/sumor copied TO :co:
-
```

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 MYTEST and a file named DIFF on a directory path ONE/MOR. Enter the following command:

```
-copy mytest/alpha,one/mor/diff <CR>
aaaaa
aaaaa
mytest/alpha copied TO :co:
yyyyy
yyyyy
one/more/diff copied TO :co:
-
```

5.7.5 Listing Directories

Previous examples have shown you how to list the contents of data files by specifying a directory pathname in a COPY command. However, you should not use the COPY command to list the contents of directories, because COPY lists the directory as though it were a data file. Instead, use the DIR command to list the directory's catalog of files as follows:

```
-dir mytest <CR>
01  JAN 87  00:00:00
DIRECTORY OF mytest  ON VOLUME disk2
alpha       beta        gamma   URTEST

-
```

See the DIR description in Chapter 4 for examples of the available listing formats.

5.7.6 Moving Files Between Directories

There may be situations when you wish to reorganize a large group of existing files under new headings (directories). You can copy files from one directory to another by using the COPY command. For example, assume you wish to copy files ALPHA, BETA, and GAMMA from your default directory to the existing directory MYTEST. Enter the following command line, using the QUERY parameter (optional):

```
-COPY alpha,beta,gamma to MYTEST QUERY <CR>
alpha, COPY TO MYTEST/alpha? y <CR>
alpha COPIED TO MYTEST/alpha
beta, COPY TO MYTEST/beta? y <CR>
beta COPIED TO MYTEST/beta
gamma, COPY TO MYTEST/gamma? y <CR>
gamma COPIED TO MYTEST/gamma
-
```

Assume you later decide to move file ALPHA back to your default directory. You need not specify the default directory in the new pathname for ALPHA. Enter the following command:

```
-rename mytest/alpha to alpha <CR>
mytest/alpha renamed TO alpha
-
```

Any subsequent operations involving file ALPHA would only require the file name. For example:

```
-copy alpha <CR>
aaaaa
aaaaa
alpha copied TO :CO:
-
```

5.7.7 Deleting 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 is not empty. For example, assume your default directory contains files FILE1, FILE2, FILE3, and directory MYDIR containing the file NODEL. Now suppose you want to delete the files in your default directory, but you accidentally enter:

```
-
```

The DELETE command will only delete the files but not the directory MYDIR because it is not empty. You will see the following display on your screen:

```
file1, deleted
file2, deleted
file3, deleted
mydir, directory not empty
```

At this point you should list the MYDIR directory by using the DIR command to determine the contents of MYDIR, as follows:

```
-dir mydir <CR>

01  JAN 87  00:00:00
DIRECTORY OF mydir  ON VOLUME disk2
NODEL

-
```

You now have two options. You can use the RENAME or COPY commands to move any files to be saved to a different directory, or you can use the DELETE command to delete the entire contents of MYDIR before deleting the directory.

Assume you wish to move NODEL to the NUTEST directory so that MYDIR itself can be deleted. Enter the following commands:

```
-rename mydir/nodel to nutest/nodel <CR>
mydir/nodel renamed TO nutest/nodel
-delete * <CR>
mydir deleted
-
```

The RENAME command automatically changes the NODEL pathname from the MYDIR directory to the nutest directory, making MYDIR empty.

5.7.8 Changing Your Default Directory

Suppose your default directory contains a directory called MYTEST which contains another directory called URTEST which in turn contains several data files called MOR, SUMOR, STILMOR, and NOMOR. If you plan to manipulate these data files extensively, your Human Interface commands can become very cumbersome, due to the length of the pathnames involved.

For example, suppose you wish to copy the data files to files called ALPHA, BETA, DELTA, and GAMMA in the same directory. The command to do this is

```
-copy mytest/urtest/mor, mytest/urtest/sumor,  &  <CR>
**mytest/urtest/stilmor, mytest/urtest/nomor  to &  <CR>
**mytest/urtest/alpha, mytest/urtest/beta, &  <CR>
**mytest/urtest/delta, mytest/urtest/gamma  <CR>
```

If there are more levels in the directory structure, your commands can become even longer.

To eliminate some of these long pathnames, you can use the ATTACHFILE command to change your default directory to be a directory closer to the level of the files with which you are working. To make the previous command shorter, you could change your default directory to the URTEST directory, as follows:

```
-attachfile mytest/urtest <CR>
mytest/urtest attached AS :$:
```

Now, when you make references to files without specifying the entire pathname, the Human Interface assumes that they reside in the URTEST directory, not your previous default directory. Therefore, to perform the same operation as in the previous COPY command, you could now enter the following command:

-

You can use the ATTACHFILE command to change your default directory to any directory that you wish, so that you can manipulate the files in that directory more easily. To return to your original default directory, enter the following command:

```
-attachfile <CR>
:HOME:,  attached AS :$:
```

This command uses the default parameters and has the same effect as "ATTACHFILE :HOME: AS :$:". The :HOME: logical name represents your original default directory; therefore the command returns :$: to its original value.

If you use several directories at one time, you may want to use the ATTACHFILE command to assign short logical names to these directories and thus, reduce the length of the pathname you need to enter each time you specify a directory. For example, you may assign the logical name :MY: to the MYDIR/NEWDIR directory as follows:

```
-attachfile mydir/newdir as :my: <CR>
mydir/newdir attached AS :MY:
```

From now until you logoff you can refer to files in the MYDIR/NEWDIR directory with the logical name :MY: as the prefix.

5.8 RENAMING FILES AND DIRECTORIES

The most direct method to save the contents of a file or directory but change its pathname is to use the RENAME command. To make the process easier to follow, this section discusses the renaming of files and directories separately.

5.8.1 Renaming 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 <CR>
alpha renamed TO omega
-
```

The ALPHA pathname is automatically deleted from the system when the RENAME command is executed. You can also rename lists of files to new pathnames In this case, it is useful to include the QUERY parameter in your command line to make certain that your old pathnames and new pathnames are matched up in the way you intend.

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 <CR>
alpha renamed TO tom
beta renamed TO dick
gamma renamed TO harry
-
```

Remember that when using the RENAME command, you must always have a one-for-one match of pathnames between the new list and the old file list. For example, more old pathnames than new pathnames would cause the following exchange at the terminal:

```
-rename alpha,beta to tom <CR>
alpha renamed TO tom
TOM, already exists, DELETE?
-
```

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

```
-rename alpha to beta,tom <CR>
008B: E$UNMATCHED LISTS
-
```

So far, these RENAME examples have used the TO parameter to give new names to existing files. However, you can also use the OVER preposition with RENAME. The primary purpose of OVER is to move data from one named file over the data in another existing file. This use of the OVER preposition matches the action of the OVER preposition in the COPY command with one important distinction: RENAME automatically deletes the input file when the command is executed.

Be careful here! It's easy to get into semantic confusion when using the OVER preposition in a RENAME command. Just remember a few simple rules:

- Use the pathname of the data to be moved to a different but existing pathname as the input parameter; that is, on the left-hand side of the OVER preposition. This pathname will be deleted when the command is executed.
- Use the pathname that receives the input data as the output parameter; that is, on the right-hand side of the OVER preposition. The previous contents of this file will be replaced when the command is executed.

For example, assume you have a file named ABLE whose contents consist of the data line aaaaa, and another file named BAKER whose contents consist of the data line bbbbb. You wish to rename ABLE with the name BAKER. Enter the following command:

```
-rename able over baker <CR>
able renamed OVER baker
-
```

Now display the contents of the file previously named ABLE but now named BAKER:

```
-copy baker <CR>
aaaaa
baker copied TO :CO:
-
```

The previous contents of BAKER have been deleted, and pathname ABLE has been deleted from its directory. You can also use the TO preposition to rename files with other existing pathnames. Using TO might be slightly less confusing but you must enter extra keystrokes. For example, assume you wish to rename ALPHA and BETA with the existing file names GAMMA and DELTA. Enter the following command:

```
-rename alpha,beta to gamma,delta <CR>
gamma, already exists, DELETE? <CR>
alpha renamed TO gamma
delta, already exists, DELETE? y <CR>
beta renamed TO delta
-
```

5.8.2 Renaming Directories

A directory can be renamed to a new pathname on the same volume (but not to an existing pathname). Assume you have a directory whose pathname is ALPHA/BETA and you wish to rename it with a new pathname of ALPHA/BEE. Enter the following command:

```
-rename alpha/beta to ALPHA/BEE <CR>
alpha/beta renamed TO ALPHA/BEE
-dir alpha/beta <CR>
alpha/beta, does not exist
-
```

Be careful when renaming directories! The last message explains the consequences of renaming a directory to a new pathname. ONCE YOU RENAME A DIRECTORY, ALL FILES LISTED UNDER THAT DIRECTORY WILL ALSO HAVE THEIR PATHNAMES CHANGED. If your system has other programs that use data files that are listed under the old directory name, those programs will never find the files. In such a case, you must either rename the directories to their original names or modify the programs.

In summary, the distinctions between using the RENAME and COPY commands are as follows:

- When you use COPY to move the contents of an existing file TO a new file or OVER an existing file, the input file still exists.
- When you use RENAME to move the contents of an input file TO a named new file or OVER an existing file, the input pathname is automatically deleted.

5.9 MOVING FILES ACROSS VOLUME BOUNDARIES

You can use all Human Interface file-handling commands except RENAME to manipulate files across volume boundaries. That is, you can copy files or directories from one diskette or disk platter to another one mounted on a different drive. The restriction against using RENAME across volume boundaries is intended for the protection of files against accidental deletion.

You access a different volume by entering the logical name for the device (the drive on which the volume is mounted) as the first item in the pathname. For example, assume you have a volume mounted on a drive whose logical name is :f1:. 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 :f1: <CR>

01 JAN 86  00:00:00
DIRECTORY OF  :f1:  ON VOLUME disk2
able        baker       chuck       OMNI        samp
BUS         nusamp      STATS

-
```

Assume you wish to copy file ABLE from this volume mounted on :f1: to the MYTEST directory (which resides in your default directory). Enter the following command:

```
-copy :f1:able to mytest/able <CR>
:f1:able copied TO mytest/able
-
```

If you then wish to delete files ABLE and BAKER from the :f1: volume, simply enter the command:

```
-delete :f1:able,:f1:baker <CR>
:f1:able, deleted
:f1:baker, deleted
-
```

Now, assume the following conditions:

- You have two data files on the :f1: 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 :f1:stats/sales/feb, :f1:stats/sales/mar  &  <CR>
**to mytest/peek/subtot <CR>
:f1:stats/sales/feb copied TO mytest/peek/subtot
:f1: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.

5.10 FORMATTING A NEW VOLUME

Whenever you wish to use a new volume on a secondary storage device (such as 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 place a new 5.25-inch diskette in a disk drive and attach it with the logical name :d:, which you have attached with the ATTACHDEVICE command as a named device.

Enter the following command:

```
-format :d: <CR>
    volume () will be formatted as a NAMED volume
    granularity          = 512          map start =301
    interleave           =   5          sides     = 2
    files                = 200          density   = double
    extensionsize        =   3          disk size = mini (5.25")
    save area reserved   = no
    bad track/sector information written = no
    volume size          = 318K

volume formatted
-
```

This formatting example exercised all the default options. It did not specify a volume name as a parameter of FORMAT. A volume name is not required; however, for diskettes, a volume name gives you a method for identifying a volume in case the stick-on label on the diskette gets lost or destroyed. You need only insert the disk in a drive and enter a DIR command for that drive to get a directory listing that specifies the volume name.

The GRANULARITY, INTERLEAVE, EXTENSIONSIZE, MAPSTART, and FILES parameters tell the FORMAT command how you want the physical space (for instance, disk surface space) on the volume allocated and accessed for maximum efficiency. The default parameters caused the example to be formatted with the following attributes:

- Since the device is attached as a named device, the NAMED parameter is the default with FORMAT. It specifies that you will be using the volume only to handle named files and directories. If you specified the PHYSICAL parameter, the entire volume would be treated as a single, large physical file. Once you define the volume as NAMED or PHYSICAL, you can only use it for that purpose.
- The GRANULARITY parameter specifies the minimum number of bytes to be allocated for each increment of file size on the volume. The default granularity is the granularity of the physical device. Once the volume granularity is defined, it is applied to every file you create on the volume.

  For example, assume the default volume granularity for your device is 1024 bytes. Each time you create a new file on the volume, the I/O System automatically allocates 1024 bytes of primary storage to that file, whether or not the file requires the full 1024 bytes. If the size of your file exceeds 1024 bytes, the I/O System will increment your file size by still another block of 1024 bytes, and so on, until the end-of-file is reached.

- The INTERLEAVE default specifies that you want an interleave factor of 5. The interleave factor defines the number of physical sectors that occur between sequential logical sectors. This value maximizes access speed for the files on a given volume, depending upon the intent of the volume and the device configuration of your system.

  For example, an interleave value of 5 for a flexible disk system means that, for each file, the I/O System will read every fifth sector on the diskette, starting from an index of 1 (other hard disk systems may be different, depending on your hardware configuration). Therefore, the I/O System does not need to wait for the disk to make a complete revolution before it accesses the next sector; the next sector by an increment of 5 is ready to be accessed for read/write by the time the first accessed sector has been processed.

  Note that the INTERLEAVE is the only optional parameter that is meaningful for volumes formatted for PHYSICAL files; the FILES, EXTENSIONSIZE, and GRANULARITY options are ignored in FORMAT commands that specify a PHYSICAL file format for the volume.

- The default FILES parameter specifies that you wish to create a maximum of 200 user files on the volume. Although the actual number of files you can specify is 1 through 65,528, at a practical level, one of your determining factors will be the incremental file size you specify in the GRANULARITY parameter.

- The default EXTENSIONSIZE parameter specifies that you wish to create three bytes of extension data for each file. The Human Interface requires that at least three bytes of extension data be available. Other system programs included in your system may require larger values.

- The MAPSTART gives the volume block number where the fnode and map files start. If you do not specify a number, the Human Interface places the fnode and map files in the center of the volume.

5.10.1 iRMX® - iNDX Compatible Diskettes

It is possible to format a diskette with iRMX II that can be read by the iNDX Operating System. Use the following parameters when attaching and formatting the diskette:

-
-

5.11 DISKETTE SWITCHING PROCEDURES

If your system contains 5.25-inch diskette drives, you might need to perform special procedures when switching diskettes.

Most 5.25-inch diskette drives are unable to signal the operating system when an operator changes diskettes. Therefore, you must always perform the following steps when changing diskettes:

1. Before removing the first diskette, invoke the DETACHDEVICE command.
2. Remove the first diskette and insert the replacement diskette.
3. Invoke the ATTACHDEVICE command (or an alias) to gain access to the new diskette.

If you don't follow this procedure, the operating system assumes that the first diskette is still in the drive. At some point it will update the diskette's directory from the information it maintains in memory. Because the directory information it writes to disk applies to the previous diskette, all the directory entries and file pointers will be wrong, causing the diskette to be unusable.

CHAPTER 6
TERMINAL SUPPORT CODE

6.1 INTRODUCTION

Every terminal connected to an iRMX II application system must be able to communicate with the system. As stated in Chapter 1, the initial program provides that means of communication. The iRMX II Human Interface provides a standard CLI with line-editing features as its initial program. However, systems that do not include the standard CLI or a standard CLI with user extensions (see the *Extended iRMX II Human Interface User's Guide*) can communicate with the system and gain access to line-editing features by using the Terminal Support Code feature of the Basic I/O System.

The Terminal Support Code is a software package that interfaces to terminal device drivers to provide terminal communication for systems that include the Basic I/O System. It is available to all application programs and can be used with all the Human Interface commands. However, some of the Terminal Support Code features cannot be used when you are issuing CLI commands.

The Terminal Support Code provides a type-ahead feature and a set of line-editing and control characters that give you the basic editing and control functions you need when entering text at a terminal. You can use these characters in addition to the Human Interface commands. This chapter discusses, along with the Terminal Support Code, the line-editing features and control characters which are available. However, the Terminal Support Code contains many features other than those discussed in this chapter. Refer to the *Extended iRMX II Device Drivers User's Guide* for information about the other features of the Terminal Support Code.

6.2 TYPE-AHEAD

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

(For an explanation of how the CLI uses this feature, see Chapter 3.)

6.3 CONTROLLING INPUT FROM A TERMINAL

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

CONTROL-character

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

The control characters listed in this section are the default characters. Each can be replaced with a different character by means of a selection procedure described in the *Extended iRMX II Device Drivers User's Guide*. The default editing and control characters for terminal input include:

CARRIAGE RETURN or LINE FEED	Terminates the current line and positions the cursor at the beginning of the next line. Entering either of these characters adds a carriage return/line feed pair to the input line.
RUBOUT	Deletes (or rubs out) the previous character in the input line. In response to the RUBOUT, your terminal display changes in one of two ways, depending on the configuration of the Terminal Support Code. In one configuration, each RUBOUT removes a character from the screen and moves the cursor back to that character position. In the other configuration, each RUBOUT echoes the deleted character back to the terminal. In the second configuration, also called hard-copy mode, the Terminal Support Code surrounds the echoed characters with the "#" character to distinguish the echoed characters from the surrounding text.

CONTROL-p	A "quoting" character, which removes, from a control character that immediately follows it, any meaning that is special to the Terminal Support Code. CONTROL-p causes the next character to become a literal, causing it to be sent on to the operating system, even if it is an input control character that the Terminal Support Code understands. All input control characters sent to the operating system in this manner are not processed as control characters. Output control characters (such as CONTROL-s, and CONTROL-q) perform their special functions even if preceded by a CONTROL-p. The CONTROL-p does not echo at the terminal.
CONTROL-r	If the current input line is not empty, this character reprints the line with editing already performed. This control character enables you to see the effects of the editing characters entered since the most recent line terminator. If the current line is empty, this character reprints the previous line, up to the point of the line terminator. Additional CONTROL-r characters display previous lines, until there are no more lines in the type-ahead buffer. Subsequent CONTROL-r characters display the last line found. This use of CONTROL-r is a convenient way to repeat a previously-entered command.
CONTROL-u	Discards the entire contents of the type-ahead buffer.
CONTROL-x	Discards the current input line. This character echoes the "#" character, followed by a carriage return/line feed, at the terminal.
CONTROL-z	If entered as the only character in a line, this character specifies an end-of-file, terminating a read from the terminal. If entered on a non-empty line, it terminates the line without appending a carriage return/line feed pair to the line.

6.4 CONTROLLING OUTPUT TO A TERMINAL

The following section applies to both the standard CLI and the Terminal Support Code. The modes and control characters described here are recognized by the standard CLI and are used in the same way as described for the Terminal Support Code.

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

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

The following control characters, when entered at the terminal, change the output mode for the terminal. Like the input control characters, these are defaults. They can be changed by a selection process described in the *Extended iRMX II Device Drivers User's Guide*.

CONTROL-o	Places the terminal in discarding mode if the terminal is in a mode other than discarding mode. If the terminal is already in discarding mode, the CONTROL-o character returns the terminal to its previous output mode.
CONTROL-q	Resumes previous output mode. If you enter this character after stopping output with the CONTROL-s character, output continues in the same manner as before you entered the CONTROL-s (that is, if your terminal was in scrolling mode before you entered CONTROL-s, output resumes in scrolling mode). Entering CONTROL-q at any other time places your terminal in normal mode (that is, all output is displayed at the terminal without waiting for permission to continue). Therefore, you can use CONTROL-q to reverse the effect of a CONTROL-w and get your terminal out of scrolling mode.

CONTROL-s	Places the terminal in stopped mode (stops output). You can resume output without loss of data by entering the CONTROL-q character. If the terminal is in discarding mode (as a result of a CONTROL-o character), the CONTROL-s character has no effect on output.
CONTROL-t	Places the terminal in scrolling mode and sets the scroll count to one. This means that you must enter another CONTROL-t character after each displayed line in order to continue the display.
CONTROL-w	Places the terminal in scrolling mode. In this mode, the terminal displays output 18 lines at a time (usually, 18 lines fills 2/3 of the the screen) and then waits for user input to continue. When you enter another CONTROL-w character, the terminal displays the next screen of information. The scrolling count is selectable; refer to the extended iRMX II BASIC I/O USER'S GUIDE for more information.

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

CONTROL-c	Flushes the type-ahead buffer and causes the operating system to abort the currently-executing program. If you enter a Human Interface command to initiate a program, you can generally enter CONTROL-c to stop it.

For an overview of the control characters, refer to Table 6-1.

Table 6-1. Overview of Default Control Characters

Characters	Results
	Default Input Control Characters
carriage return or line feed	terminates current line and puts cursor at start of next line
rubout	deletes single character
CONTROL-p	removes any special meaning from input control characters except CONTROL-c; has no affect on output control characters
CONTROL-r	reprints line
CONTROL-u	flushes type-ahead buffer
CONTROL-x	discards current input line
CONTROL-z	specifies an end of file
	Default Output Control Characters
CONTROL-o	places terminal in discarding mode
CONTROL-q	resumes output mode
CONTROL-s	stops output
CONTROL-t	scrolls output one line at a time
CONTROL-w	scrolls output one screen at a time
CONTROL-c	aborts currently executing program

6.5 ESCAPE SEQUENCES

The Terminal Support Code also accepts escape characters that allow you to further define your terminal. (For example, you could set the scroll count or switch your terminal into transparent mode so that control characters have no effect.) You can enter these escape characters from the terminal, or you can write them to the terminal from a program. For information about these escape codes, refer to the *Extended iRMX II Device Drivers User's Guide*.

NOTE

This feature, like many of the other features of Terminal Support Code, does not apply when using the Human Interface CLI as your initial program.

APPENDIX A
CONDITION CODES

A.1 INTRODUCTION

This appendix provides a complete list of the iRMX II condition codes that can occur during system operations. The condition codes are divided into two categories:

- environmental conditions
- programmer errors

A programmer error is a condition that your program can prevent whereas, an environmental condition is caused by a problem in the operating system that you cannot control.

Table A-1 lists the condition codes by layer with their numeric values and mnemonics.

In addition, a separate section at the end of this appendix lists condition codes that exist in iRMX I.7, but do not exist in iRMX II.2 or iRMX II.3.

Table A-1. Conditions And Their Codes

Category/Mnemonic	Hex Code	Decimal Code	Description
E$OK	0H	0	The last system call that returned a status was successful.
Nucleus Environmental Conditions			
E$TIME	01H	1	A time limit (possibly a limit of zero time) expired without a task's request being satisfied.
E$MEM	02H	2	There is not sufficient memory available to satisfy a task's request.
E$BUSY	03H	3	Another task currently has access to the data protected by a region.
E$LIMIT	04H	4	A task attempted an operation which, if it had been successful, would have violated a Nucleus-enforced limit.
E$CONTEXT	05H	5	A system call was issued out of context or the operating system was asked to perform an impossible operation.
E$EXIST	06H	6	A token parameter has a value which is not the token of an existing object.
E$STATE	07H	7	A task attempted an operation which would have caused an impossible transition of a task's state.
ENOTCONFIGURED	08H	8	This system call is not part of the present configuration.
E$INTERRUPT$SATURATION	09H	9	An interrupt task has accumulated the maximum allowable number of SIGNAL$INTERRUPT requests.

Table A-1. Conditions And Their Codes (continued)

Category/Mnemonic	Hex Code	Decimal Code	Description
E$INTERRUPT$OVERFLOW	0AH	10	An interrupt task has accumulated more than the maximum allowable amount of SIGNAL$INTERRUPT requests.
E$TRANSMISSION	0BH	11	A NACK, timeout, or bus error occurred.
E$SLOT	0CH	12	There are no available GDT slots.
E$DATA$CHAIN	0DH	12	A data chain has been returned. The TOKEN points to the beginning of the data chain block.
I/O System Environmental Conditions			
E$FEXIST	20H	32	The specified file already exists.
E$FNEXIST	21H	33	The specified file does not exist.
E$DEVFD	22H	34	The device driver and file driver are incompatible.
E$SUPPORT	23H	35	The combination of parameters entered is not supported.
E$EMPTY$ENTRY	24H	36	The specified entry in a directory file is empty.
EDIREND	25H	37	The specified directory entry index is beyond the end of the directory file.
E$FACCESS	26H	38	The connection does not have the correct access to the file.
E$FTYPE	27H	39	The requested operation is not valid for this file type.

Table A-1. Conditions And Their Codes (continued)

Category/Mnemonic	Hex Code	Decimal Code	Description
E$SHARE	28H	40	The requested operation attempted an improper kind of file sharing.
E$SPACE	29H	41	There is no space left on the volume.
E$IDDR	2AH	42	An invalid device driver request occurred.
E$IO	2BH	43	An I/O error occurred.
E$FLUSHING	2CH	44	The connection specified in the call was deleted before the operation completed.
E$ILLVOL	2DH	45	The device contains an invalid or improperly-formatted volume.
EDEVOFFLINE	2EH	46	The device being accessed is now offline.
E$IFDR	2FH	47	An invalid file driver request occurred.
E$FRAGMENTATION	30H	48	The volume is too fragmented for a file to be extended.
EDIRNOT$EMPTY	31H	49	The call is attempting to delete a directory that is not empty.
ENOTFILE$CONN	32H	50	The connection parameter is a device connection, not a file connection.
ENOTDEVICE$CONN	33H	51	The connection parameter is not a device connection.
E$CONN$NOT$OPEN	34H	52	The connection is not open for reading, writing or updating.

Table A-1. Conditions And Their Codes (continued)

Category/Mnemonic	Hex Code	Decimal Code	Description
E$CONN$OPEN	35H	53	The task attempted to open a connection that is already open.
E$BUFFERED$CONN	36H	54	The specified connection was opened by the EIOS, and used by the BIOS which is not allowed. Once you have an open connection, you must manipulate it with a system call provided by the same I/O System.
E$OUTSTANDING$CONNS	37H	55	A soft detach was specified, but connections to the device still exist.
E$ALREADY$ATTACHED	38H	56	The specified device is already attached.
EDEVDETACHING	39H	57	The file specified is on a device that the operating system is in the process of detaching.
ENOTSAME$DEVICE	3AH	58	The existing pathname and the new pathname refer to different devices. You cannot simultaneously rename a file and move it to another device.
E$ILLOGICAL$RENAME	3BH	59	The call is attempting to rename a directory to a new path containing itself.
E$STREAM$SPECIAL	3CH	60	A stream file request is out of context. Either it is a query request and another query request is already queued, or it is a satisfy request and either the request queue is empty or a query request is queued.
E$INVALID$FNODE	3DH	61	The connection refers to a file with an invalid fnode. You should delete this file.
E$PATHNAME$SYNTAX	3EH	62	The specified pathname contains invalid characters.

Table A-1. Conditions And Their Codes (continued)

Category/Mnemonic	Hex Code	Decimal Code	Description
E$FNODE$LIMIT	3FH	63	The volume already contains the maximum number of files. No more fnodes are available for new files.
ELOGNAME$SYNTAX	40H	64	The specified pathname starts with a colon (:), but it does not contain a second, matching colon; the specified pathname has more than 12 characters or contains invalid characters.
E$CANNOT$CLOSE	41H	65	The buffers cannot be written to the device to complete the I/O request.
E$IOMEM	42H	66	The Basic I/O System has insufficient memory to process a request.
E$MEDIA	44H	68	The device containing a specified file is not on-line.
ELOGNAME$NEXIST	45H	69	The specified path contains an explicit logical name, but the Extended I/O System was unable to find the name in the object directories of the local job, the global job, and the root job.
ENOTOWNER	46H	70	The user who attempted to detach the device is not the owner of the device.
EIOJOB	47H	71	The Extended I/O System cannot create an I/O job because the size specified for the object directory is too small.
EUDFFORMAT	48H	72	The User Definition File is not in the right format.
E$NAME$NEXIST	49H	73	The user name specified in the call is not listed in the User Definition File.

Table A-1. Conditions And Their Codes (continued)

Category/Mnemonic	Hex Code	Decimal Code	Description
EUIDNEXIST	4AH	74	The user ID in the specified user object does not match the ID listed in the User Definition File for the corresponding user name.
E$PASSWORD$MISMATCH	4BH	75	The password specified in the call does not match the one listed in the User Definition File for the corresponding user name.
EUDFIO	4CH	76	The User Definition File specified cannot be found. This helps you know when an error code came from a remote UDF and not another remote file.
EIOUNCLASS	50H	80	An unknown type of I/O error occurred.
EIOSOFT	51H	81	A soft I/O error occurred. A retry might be successful.
EIOHARD	52H	82	A hard I/O error occurred. A retry is probably useless.
EIOOPRINT	53H	83	The device was off-line. Operator intervention is required.
EIOWRPROT	54H	84	The volume is write-protected.
EIONO$DATA	55H	85	A tape drive attempted to read the next record, but it found no data.
EIOMODE	56H	86	A tape drive attempted a read/write operation before the previous write (read) completed.

Table A-1. Conditions And Their Codes (continued)

Category/Mnemonic	Hex Code	Decimal Code	Description
Application Loader Environmental Conditions			
EBADHEADER	62H	98	The object file contains an invalid header record.
E$EOF	65H	101	The Application Loader encountered an unexpected end-of-file while reading a record.
ENOLOADER$MEM	67H	103	There is insufficient memory to satisfy the memory requirements of the Application Loader.
ENOSTART	6CH	108	The Application Loader could not find the start address.
EJOBSIZE	6DH	109	The maximum memory-pool size of the job being loaded is smaller than the amount of memory required to load its object file.
E$OVERLAY	6EH	110	The overlay name does not match any of the overlay module names.
E$LOADER$SUPPORT	6FH	111	The file requires features not supported by the Application Loader as configured.
Human Interface Environmental Conditions			
E$LITERAL	80H	128	The parsing buffer contains a literal with no closing quote.
E$STRING$BUFFER	81H	129	The string to be returned exceeds the size of the buffer the user provided in the call.
E$SEPARATOR	82H	130	The parsing buffer contains a command separator.

Table A-1. Conditions And Their Codes (continued)

Category/Mnemonic	Hex Code	Decimal Code	Description
E$CONTINUED	83H	131	The parse buffer contains a continuation character.
E$INVALID$NUMERIC	84H	132	A numeric value contains invalid characters.
E$LIST	85H	133	A value in the value list is missing.
E$WILDCARD	86H	134	A wild-card character appears in an invalid context, such as in an intermediate component of a pathname.
E$PREPOSITION	87H	135	The command line contains an invalid preposition.
E$PATH	88H	136	The command line contains an invalid pathname.
E$CONTROL$C	89H	137	The user typed a CONTROL-C to abort the command.
E$CONTROL	8AH	138	The command line contains an invalid control.
E$UNMATCHED$LISTS	8BH	139	The number of files in the input and output pathname lists is not the same.
E$INVALID$DATE	8CH	140	The operator entered an invalid date.
ENOPARAMETERS	8DH	141	A command expected parameters, but the operator didn't supply any.
E$VERSION	8EH	142	The Human Interface is not compatible with the version of the command the operator invoked.
EGETPATH$ORDER	8FH	143	A command called CGETOUTPUT$PATHNAME before calling CGETINPUT$PATHNAME.

Table A-1. Conditions And Their Codes (continued)

Category/Mnemonic	Hex Code	Decimal Code	Description
E$PERMISSION	90H	144	The user does not have permission to access the requested resource.
E$INVALID$TIME	91H	145	The operator entered an invalid time.
UDI Environmental Conditions			
E$UNKNOWN$EXIT	0C0H	192	The program exited normally.
E$WARNING$EXIT	0C1H	193	The program issued warning messages.
E$ERROR$EXIT	0C2H	194	The program detected errors.
E$FATAL$EXIT	0C3H	195	A fatal error occurred in the program.
E$ABORT$EXIT	0C4H	196	The operating system aborted the program.
EUDIINTERNAL	0C5H	197	A UDI internal error occurred.
Communications System Environmental Conditions			
E$CANCELLED	0E1H	225	A SEND$RSVP transaction has been remotely canceled.
E$HOST$ID	0E2H	226	The host$id portion of the socket parameter does not refer to an agent (board) that is currently in the message space.

Table A-1. Conditions And Their Codes (continued)

Category/Mnemonic	Hex Code	Decimal Code	Description
ENOLOCAL$BUFFER	0E3H	227	This error applies in the following two cases: 1) If the receive$type parameter indicates a request message, the local port's buffer pool does not contain a buffer large enough to hold the message so the RQ$RECEIVE$FRAGMENT system call is required (message fragmentation). 2) If the receive$type parameter indicates a response message, the RSVP buffer supplied in the RQ$SEND$RSVP system call is not large enough to hold the response.
ENOREMOTE$BUFFER	0E4H	228	The remote port's buffer pool does not have a buffer large enough to hold the message and message fragmentation is turned off.
E$RESOURCE$LIMIT	0E6H	230	Either the number of simultaneous messages, or simultaneous transactions, has been reached. These fields are set during system configuration.
E$TRANS$ID	0E8H	232	The specified transaction ID is not valid. This is the rsvp$trans$id that was sent in the initial SEND$RSVP call.
E$DISCONNECTED	0E9H	233	The port sending the message has previously issued an RQ$CONNECT to a remote port. The board on which the remote port is located has been reset.
E$TRANS$LIMIT	0EAH	234	A transmission resource limitation has been encountered.
Nucleus Programmer Errors			
E$ZERO$DIVIDE	8000H	32768	A task attempted a divide in which the quotient was larger than 16 bits.

Table A-1. Conditions And Their Codes (continued)

Category/Mnemonic	Hex Code	Decimal Code	Description
E$OVERFLOW	8001H	32769	An overflow interrupt occurred.
E$TYPE	8002H	32770	A token referred to an existing object that is not of the required type.
E$PARAM	8004H	32772	A parameter that is neither a token nor an offset has an invalid value.
EBADCALL	8005H	32773	An OS extension received an invalid function code.
E$ARRAY$BOUNDS	8006H	32774	Hardware or software has detected an array overflow.
ENDPERROR	8007H	32775	A Numeric Processor (NPX) error has occurred. OS extensions can return the status of the NPX to the exception handler.
E$ILLEGAL$OPCODE	8008H	32776	The processor tried to execute an invalid instruction.
E$EMULATOR$TRAP	8009H	32777	An ESC instruction was encountered with the emulator bit set in the machine status word.
E$CHECK$EXCEPTION	800AH	32778	A Pascal task has exceeded the bounds of a CASE statement.
ECPUXFER$DATA$LIMIT	800BH	32779	The NPX tried to access an address that is out of segment boundaries.
E$PROTECTION	800DH	32781	A general protection error.
ENOTPRESENT	800EH	32782	A request has been made to load a segment register whose segment is not present.

Table A-1. Conditions And Their Codes (continued)

Category/Mnemonic	Hex Code	Decimal Code	Description
EBADADDR	800FH	32783	The logical address is illegal. Either the selector does not point to a valid segment, or the offset is not within the segment boundaries.
I/O System Programmer Errors			
E$NOUSER	8021H	32801	No default user is defined.
E$NOPREFIX	8022H	32802	No default prefix is defined.
EBADBUFF	8023H	32803	Illegal usage of memory buffers in read or write requests.
ENOTLOG$NAME	8040H	32832	The specified object is not a device connection or file connection.
ENOTDEVICE	8041H	32833	A token parameter referred to an existing object that is not, but should be, a device connection.
ENOTCONNECTION	8042H	32834	A token parameter referred to an existing object that is not, but should be, a file connection.
Application Loader Programmer Error			
EJOBPARAM	8060H	32864	The maximum memory pool size specified for the job is less than the minimum pool size specified.

Table A-1. Conditions And Their Codes (continued)

Category/Mnemonic	Hex Code	Decimal Code	Description
Human Interface Programmer Errors			
E$PARSE$TABLES	8080H	32896	There is an error in the internal parse tables.
EJOBTABLES	8081H	32897	An internal Human Interface table was overwritten, causing it to contain an invalid value.
E$DEFAULT$SO	8083H	32899	The default output name string is invalid.
E$STRING	8084H	32900	The pathname to be returned exceeds 255 characters in length.
E$ERROR$OUTPUT	8085H	32901	The command invoked by C$SEND$COMMAND includes a call to C$SEND$EO$RESPONSE, but the command connection does not permit C$SEND$EO$RESPONSE calls.
UDI Programmer Errors			
E$RESERVE$PARAM	80C6H	32966	The calling program tried to reserve memory for more than 12 files or buffers.
E$OPEN$PARAM	80C7H	32967	The calling program requested more than two buffers when opening a file.

Table A-1. Conditions And Their Codes (continued)

Category/Mnemonic	Hex Code	Decimal Code	Description
Communication System Programmer Errors			
E$PROTOCOL	80E0H	32992	The port specified in the port$tkn parameter is of the signal type, not the data communication type.
E$PORT$ID$USED	80E1H	32993	The port$id specified for a data transaction port is in use
ENUCBAD$BUF	80E2H	32994	The info$ptr is invalid or points to a buffer that is not large enough.

A.2 OBSOLETE CONDITION CODES

The following condition codes exist in iRMX I.7, but do not exist in iRMX II.2 or iRMX II.3:

- EABSADDRESS
- EBADGROUP
- EBADSEGDEF
- E$CHECKSUM
- E$FIXUP
- EIOALT$ASSIGNED
- EIONO$SPARES
- ENOMEM
- ERELFORMAT
- ERELLENGTH
- ERELTYPE
- ESEGBOUNDS

APPENDIX B
TERMINAL SPECIFICATIONS

B.1 INTRODUCTION

The CLI is designed to run in several environments and with various terminals. The characteristics of your environment/terminal are specified with configuration commands in the :CONFIG:TERMCAP file. The parameters and control sequences that must be specified for the CLI function keys are listed in Table B-1.

The iRMX II Operating System supplies a file (:CONFIG:TERMCAP) which includes the default configuration commands with values for various terminals. The terminal types configured in the :CONFIG:TERMCAP are listed here:

Terminal Name (as in file)	Terminal Type
1510E	Hazeltine 1510 with escape lead-in
1510T	Hazeltine 1510 with tilda lead-in
ADM3A	Lear Seigler ADM-3A
QVT102	Qume QVT102, in VT102 mode
TV910P	Televideo 910 Plus
TV950	Televideo 950
VT100	Digital Equipment Corporation VT100, VT101, VT102
VT52	Digital Equipment Corporation VT52
WYSE50	Wyse 50
ZENTEC	Zentec Zepher and Cobra

In addition to the terminals listed, you may also define a terminal by the name "ANY". The "ANY" option applies to all terminal types, however, it has a limited number of functions. Its main purpose is to provide compatibility with previous iRMX II releases. It also provides a generic terminal type as a starting point.

B.2 TERMINAL CONTROL SEQUENCES

Table B-1 lists the control sequences which must be used to define the CLI function keys.

Table B-1. Control Sequences

Codes	Meaning
NAME = string	string is the terminal name
Input Codes	
AB = hhhh	Sets <ESC>
AFCL = hhhh	Sets <LEFT>
AFCR = hhhh	Sets <RIGHT>
AFCU = hhhh	Sets <UP>
AFCD = hhhh	Sets <DOWN>
AFCH = hhhh	Sets <HOME>
AR = hhhh	Sets <RUBOUT>
AFXF = hhhh	Sets delete character <DELCH>
AFXA = hhhh	Sets delete right <DELR>
AFXX = hhhh	Sets delete left <DELL>
Output Codes	
AFMB = hhhh	Moves cursor to start of line
AFML = hhhh	Moves cursor left
AFMR = hhhh	Moves cursor right
AFEK = hhhh	Erases entire line
AFEL = hhhh	Erases to the end of the line

The following legend provides an explanation of the symbols in Table B-1.

Symbol	Meaning
hhhh	One to four-byte hexadecimal number
string	iRMX II string with a maximum of 7 characters

In addition to the symbols in Table B-1, there are two delimiting characters that are used when defining the terminal specification file. These are

;	Terminal specifications separator (optional)
//	Terminal specifications terminator (after each terminal definition)

A null specification can be entered as part of any control sequence. In the case of a null specification, the CLI tries to bypass the missing output character by simulating its function. For example, if a terminal has no rubout character, you would enter:

```
AR=;
```

B.3 A SAMPLE DEFINITION FILE

The following example shows how to use the terminal definition file to define a terminal. The example assumes you are entering data from a Digital Equipment Corporation VT100 terminal. The values shown are the default values used by the CLI. Note that blank characters between control sequences are optional.

```
NAME= VT100
AB= 1B4F53;AFCL= 1B5B44;AFCR= 1B5B43;AFCU= 1B5B41;
AFCD= 1B5B42;AFCH= 1B5B50;AR= 7F;AFXF= ;AFXX= 18;
AFXA= 01; AFMB= 0D;AFML= 1B5B44;
AFMR= 1B5B43; AFEK= 1B5B324B;AFEL= 1B5B4B;//
```

In the example above, it is possible to use the PF4 function key for <ESC> (AB), and the PF1 key for <HOME> (AFCH).

If you are not sure which terminal your system will include, or if you need to be compatible with previous iRMX II releases, you may want to define a terminal using the ANY option as shown below.

```
NAME= ANY;
AB= 1B;AFCL= FF;AFCR= FF;AFCU= 12;
AFCD= 0A;AFCH= FF;AR= 7F;AFXF= FF;AFXX= FF;
AFXA= FF;
AFMB= 0D;AFML= 08;AFMR=;
AFEK= ;AFEL= ;BELL= 07;//
```

A value of FF means the function is not available on this terminal. In the above example, there is no key which performs the <HOME> function, the delete character, the delete left and the delete right functions.

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z

intel

MASTER INDEX

Intel Corporation
3065 Bowers Avenue
Santa Clara, California 95051

MASTER INDEX PREFACE

INTRODUCTION

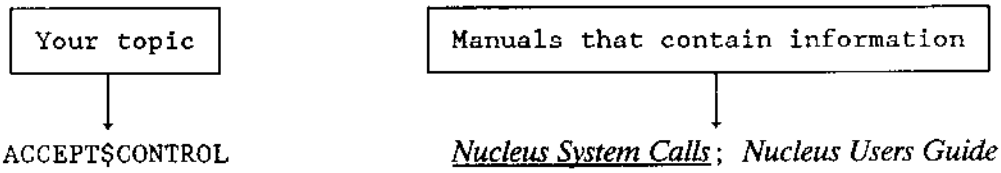
This manual contains the Master Index for the five-volume Extended iRMX II Operating System documentation set. Use this manual to locate information in the volumes.

This preface explains how to use the Master Index to find information in the manuals. The abbreviations used to identify each manual are underlined in the table below.

Manual Title	Volume
Extended iRMX II Application Loader User's Guide	(Vol. 2)
Extended iRMX II Basic I/O System User's Guide	(Vol. 2)
Extended iRMX II Bootstrap Loader Reference Manual	(Vol. 4)
Extended iRMX II Device Driver's User's Guide	(Vol. 2)
Extended iRMX II Disk Verification Utility Reference Manual	(Vol. 4)
Extended iRMX II Extended I/O System User's Guide	(Vol. 2)
Extended iRMX II Human Interface User's Guide	(Vol. 2)
Guide to the Extended iRMX II Interactive Configuration Utility	(Vol. 4)
Extended iRMX II Interactive Configuration Utility Reference Manual	(Vol. 5)
Introduction to the Extended iRMX II Operating System	(Vol. 1)
Extended iRMX II Hardware and Software Installation Guide	(Vol. 1)
Extended iRMX II Nucleus User's Guide	(Vol. 2)
Operator's Guide to the Extended iRMX II Human Interface	(Vol. 1)
Extended iRMX II Programming Techniques Reference Manual	(Vol. 4)
Extended iRMX II System Debugger Reference Manual	(Vol. 4)
Extended iRMX II Application Loader System Calls Reference Manual	(Vol. 3)
Extended iRMX II Basic I/O System Calls Reference Manual	(Vol. 3)
Extended iRMX II Extended I/O System Calls Reference Manual	(Vol. 3)
Extended iRMX II Human Interface System Calls Reference Manual	(Vol. 3)
Extended iRMX II Nucleus System Calls Reference Manual	(Vol. 3)
Extended iRMX II UDI System Calls Reference Manual	(Vol. 3)
Extended iRMX II Universal Development Interface User's Guide	(Vol. 2)

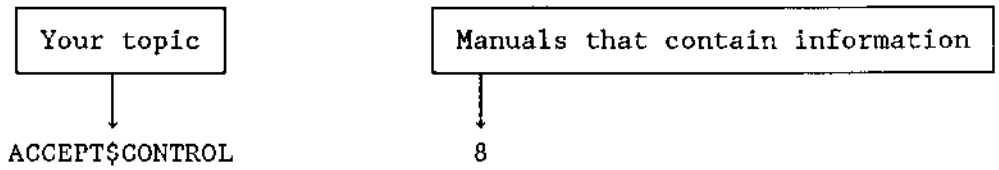
HOW TO USE THIS INDEX

This index contains an alphabetic list of topics. After locating a particular topic read the manual abbreviation to the right of your topic. Refer to the table on page iii of this index for the full name of the manual and the volume that contains it. For example:



To find this reference in the documentation, follow these steps:

1. Locate the indicated manual, in this example it is the *Extended iRMX II Nucleus System Calls Reference Manual* in Volume 3. In the case of several manuals being listed as reference, the primary source of information will be underlined.
2. After finding the correct volume, locate the particular manual that contains the information you are looking for. The same alphabetic reference will be in the manual's individual index. The primary reference contained in the manual will be underlined.



A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

INTERNATIONAL SALES OFFICES

INTEL CORPORATION
3065 Bowers Avenue
Santa Clara, California 95051

BELGIUM
Intel Corporation SA
Rue des Cottages 65
B-1180 Brussels

DENMARK
Intel Denmark A/S
Glentevej 61-3rd Floor
dk-2400 Copenhagen

ENGLAND
Intel Corporation (U.K.) LTD.
Piper's Way
Swindon, Wiltshire SN3 1RJ

FINLAND
Intel Finland OY
Ruosilante 2
00390 Helsinki

FRANCE
Intel Paris
1 Rue Edison-BP 303
78054 St.-Quentin-en-Yvelines Cedex

ISRAEL
Intel Semiconductors LTD.
Atidim Industrial Park
Neve Sharet
P.O. Box 43202
Tel-Aviv 61430

ITALY
Intel Corporation S.P.A.
Milandfiori, Palazzo E/4
20090 Assago (Milano)

JAPAN
Intel Japan K.K.
Flower-Hill Shin-machi
1-23-9, Shinmachi
Setagaya-ku, Tokyo 15

NETHERLANDS
Intel Semiconductor (Netherland B.V.)
Alexanderpoort Building
Marten Meesweg 93
3068 Rotterdam

NORWAY
Intel Norway A/S
P.O. Box 92
Hvamveien 4
N-2013, Skjetten

SPAIN
Intel Iberia
Calle Zurbaran 28-IZQDA
28010 Madrid

SWEDEN
Intel Sweden A.B.
Dalvaegen 24
S-171 36 Solna

SWITZERLAND
Intel Semiconductor A.G.
Talackerstrasse 17
8125 Glattbrugg
CH-8065 Zurich

WEST GERMANY
Intel Semiconductor G.N.B.H.
Seidlestrasse 27
D-8000 Munchen