intel®

INTELLEC® SERIES-IV OPERATING AND PROGRAMMING GUIDE

Order Number: 121753-001

AEDIT	iMMX	iPDS	MULTIBUS
BXP	Insite	iRMX	MULTICHANNEL
CREDIT	int$_e$l	iSBC	MULTIMODULE
i	int$_e$lBOS	iSBX	Plug-A-Bubble
I^2ICE	Intelevision	iSDM	PROMPT
ICE	int$_e$ligent Identifier	iSXM	Ripplemode
iCS	int$_e$ligent Programming	Library Manager	RMX/80
iDBP	Intellec	MCS	RUPI
iDIS	Intellink	Megachassis	System 2000
iLBX	iOSP	MICROMAINFRAME	UPI
i$_m$			

A987/683/ 4K DD

REV.	REVISION HISTORY	DATE
-001	Original issue.	1/83

PREFACE

This manual provides operating and programming instructions for the Intellec Series-IV Development System. The Series IV has two operating environments: one for 8086/8088-based software and one for 8080/8085-based software. Chapters 1-5 comprise the operations guide; 6 and 7 comprise the programmers reference manual.

This manual is designed to support new development system users as well as those who are already familiar with Intellec development systems. This manual assumes that you have read the *Intellec Series-IV Microcomputer Development System Overview* and are familiar with the terms listed in its glossary.

Series-IV Development System (Front View) 121753-11

This manual has seven chapters and six appendices:

- Chapter 1, "Operational Introduction," introduces you to the Series-IV Development System.
- Chapter 2, "Human Interface," describes the terminal, editing capabilities, device names, console operations, menu selection, the command line interpreter, command files, and job control functions.
- Chapter 3, "File System Management," describes the iNDX hierarchical file structure, directory files, file access and ownership, and file system considerations.
- Chapter 4, "Commands," describes the Series-IV commands and gives examples for storing, identifying, and manipulating your files and jobs.
- Chapter 5, "Using the Series-IV in the Network," describes the NDS-II and remote job control, and operating your Series IV in the NDS-II Network.
- Chapter 6, "Programming Introduction," describes operating system considerations and target environments, and lists the built-in service routines.
- Chapter 7, "The 8086/8088-Based Environment," defines the conceptual considerations and external procedures for the Series-IV service routines in an 8086/8088-based environment.
- Appendix A, "CLI Command Syntax," lists the syntax of the commands detailed in Chapter 4.
- Appendix B, "Parameters and System Service Routines," is a condensed version of the routines detailed in Chapter 7.
- Appendix C, "Error Messages and Exception Codes," lists the various error messages generated by the operating system and the UDI interface.
- Appendix D, "Object Module Relocation and Linkage," defines the possibilities for combining object modules.
- Appendix E, "Boot Device-Configuration Switch Assignments," shows the configuration switch settings necessary to boot the operating system from various physical devices.
- Appendix F, "ASCII Codes," shows ASCII codes, their meanings, and their decimal, octal, and hexadecimal values.

Required Software

The Series-IV is an Intellec Microcomputer Development System that provides support for both development and execution of programs using either the 8086/8088 chip or the 8080/8085 chip. The Series IV contains both hardware and software beyond earlier versions of Intellec development systems.

Your system contains:

iNDX —	the 8086/8088-based operating system.
ISIS-IV —	the 8080/8085-based operating system.
AEDIT™ —	an 8086/8088 screen-based text editor.
CREDIT —	an 8080/8085 screen-based text editor.
DEBUG-88 —	a low-level symbolic debugger.
MON85 —	a monitor for 8085 programs.

Related Publications

For more information on the Series-IV Microcomputer Development System, see the following manuals:

- *AEDIT™ Text Editor User's Guide*, 121756
- *Intellec Microcomputer Development System Overview*, 121752
- *Intellec Series-IV ISIS-IV User's Guide*, 121880
- *DEBUG-88 User's Guide*, 121758

Notational Conventions

UPPERCASE	Characters shown in uppercase must be entered in the order shown. You may enter the characters in uppercase or lowercase.
italic	Italic indicates a meta symbol that may be replaced with an item that fulfills the rules for that symbol. The actual symbol may be any of the following:
directory-name	Is that portion of a *pathname* that acts as a file locator by identifying the device and/or directory containing the *filename*.
filename	Is a valid name for the part of a *pathname* that names a file.
pathname	Is a valid designation for a file; in its entirety, it consists of a *directory* and a *filename*.
pathname1, pathname2, ...	Are generic labels placed on sample listings where one or more user-specified pathnames would actually be printed.
system-id	Is a generic label placed on sample listings where an operating system-dependent name would actually be printed.
V*x.y*	Is a generic label placed on sample listings where the version number of the product that produced the listing would actually be printed.
[ ]	Brackets indicate optional arguments or parameters.
{ }	One and only one of the enclosed entries must be selected unless the field is also surrounded by brackets, in which case it is optional.
{ }...	At least one of the enclosed items must be selected unless the field is also surrounded by brackets, in which case it is optional. The items may be used in any order unless otherwise noted.
|	The vertical bar separates options within brackets [ ] or braces { }.
...	Ellipses indicate that the preceding argument or parameter may be repeated.

`[,...]`	The preceding item may be repeated, but each repetition must be separated by a comma.
punctuation	Punctuation other than ellipses, braces, and brackets must be entered as shown. For example, the punctuation shown in the following command must be entered: `SUBMIT PLM86(PROGA,SRC,'9 SEPT 81')`
`input lines`	In interactive examples, user input lines are printed in white on black to differentiate them from system output.
`<cr>`	Indicates a carriage return.
shading	Shading highlights the commands that can only be used if your development system is part of the NDS-II Network (see Chapter 5).

CONTENTS

TABLES

FIGURES

CHAPTER 1
OPERATIONAL INTRODUCTION

This book will assist you in gaining hands-on experience with the Intellec Series-IV Microcomputer Development System. The Series-IV provides two execution environments and development facilities: an 8086/8088 side and an 8080/8085 side. In addition, the Series-IV has extensive facilities for teaching you to use it via a human interface that has helpful messages.

This manual is organized by topical groupings, allowing you first to acclimate yourself with the system, and then learn the theories of its operation and programming. Before reading this manual, you should read the *Series-IV Overview*, 121752, because it fully describes the features and reference library of the Series-IV. Once you learn to interface with your development system, use this guide to supplement this and other facilities of the Series-IV.

Operational Procedures

This section describes how to start-up and shut-down the Series-IV development system.

When you initialize your development system, two different diagnostics tests will be executed. The first tests the hardware (internal) power-up sequence, the second tests the software (operating system) sequence.

WARNING

Always power-up the Series IV before turning on any peripheral devices and turn off any peripherals before shutting down the Series IV.

If you are using a Winchester disk drive, do not repeatedly shut it off and power it up after booting the disk drive.

Never dismount the system volume.

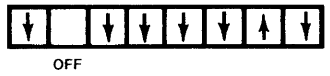
Initiating Operation

Booting From a Flexible Disk

If your Series-IV has a single flexible disk and an integrated 5-1/4″ Winchester disk drive, use these instructions the *first* time you operate your system. Once you have logged on, you can use example 2 of the "Disk Formatting Procedures" (which appear later in this chapter) to configure your system to boot from the integrated 5-1/4″ Winchester. Thereafter, when you initiate operation of your development system, follow the instructions for booting from the integrated 5-1/4″ Winchester.

To boot from a flexible disk:

1. Verify the configuration switches are set as shown below.



2. Power up the Series IV by pressing on the breaker switch located at the left rear of the terminal chassis.
3. Verify the system passed the power-up diagnostics test. If it did not, contact your Intel Service Representative.

4. Turn on any peripherals such as a Winchester disk drive or a printer. When using 740 Hard Disk Drives, press the STOP/START button and wait for the READY light before proceeding.
5. Insert the 5-1/4″ flexible operating system diskette (see figure 1-1) into the right-hand integral disk drive on the Series-IV and close the disk drive latch (see figure 1-2). The read/write access hole should face the rear of the drive and the (uncovered) write/protect slot should face the left side of the disk drive.
6. Press the RESET switch (see figure 1-3) to boot the operating system.
7. Verify the diagnostics test passed. If it did not, refer to the *Intellec Series-IV Installation and Checkout Manual.*
8. Enter the appropriate date and time as requested. The format is MM/DD/YY for the date and HH:MM:SS for the time.
9. Enter your assigned user name. If a Winchester or Hard Disk is part of your system, also enter your password when requested.

   The first time you log on after installing your system, enter the user name SUPERUSER. Your associated password is PASSME. This is your system identification until you establish other users on a shared file structure. For information on making your files and workstation accessible to others, see "File Access and Ownership" in Chapter 3 and the USERDEF and CHPASS commands in Chapter 4.

The system is now ready to accept commands. Use the facilities detailed in the *Series-IV Overview*, 121752, to assist you in learning to interface with your system as you design and develop your projects using the Series-IV.

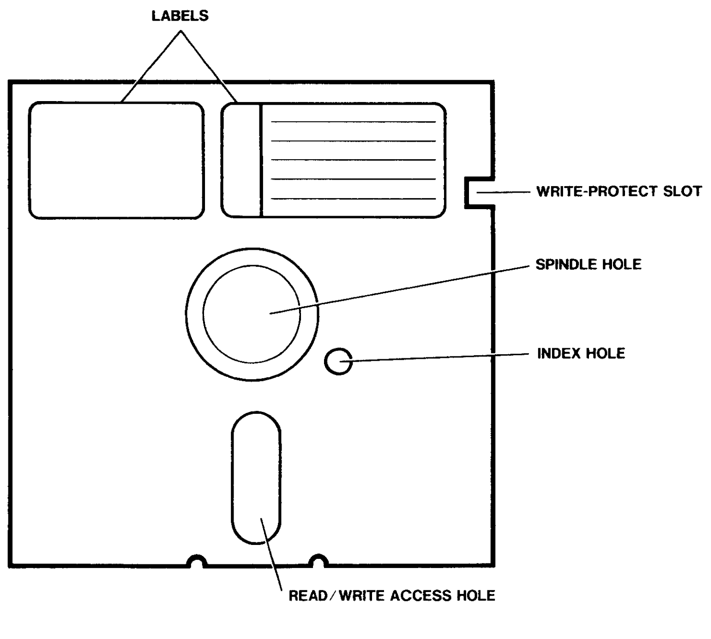


Figure 1-1. Flexible Disk 121753-1

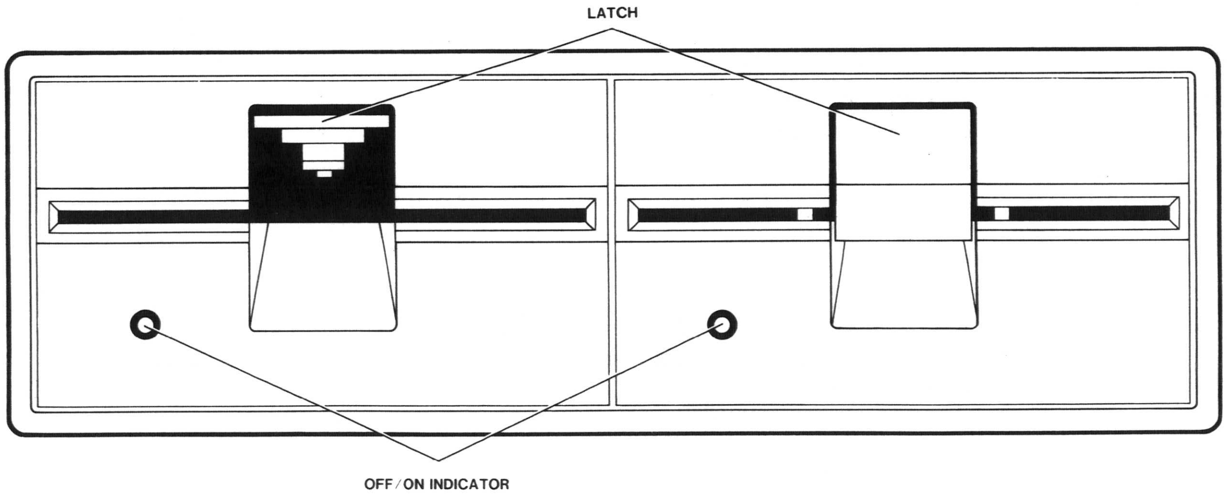


Figure 1-2. Flexible Disk Drives 121753-2

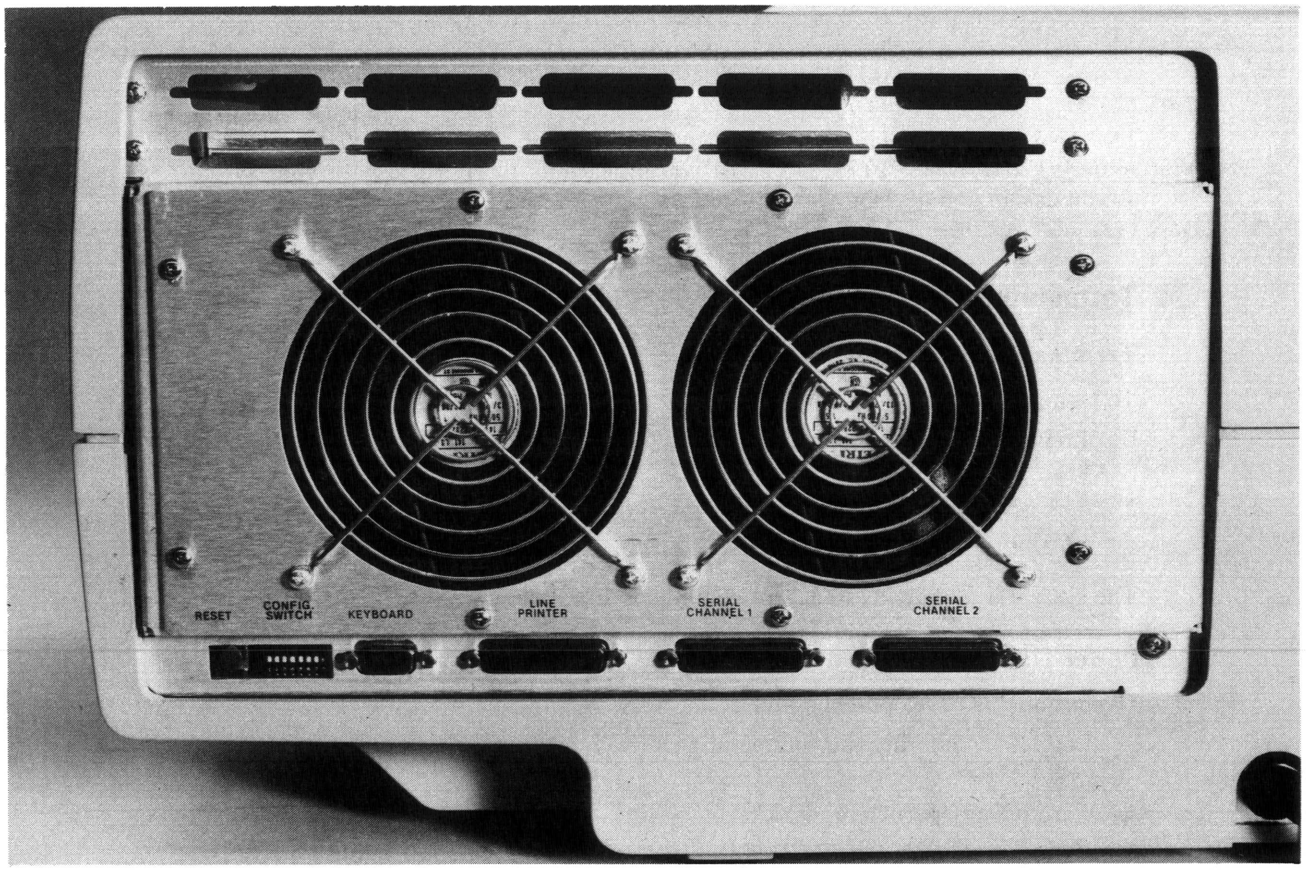
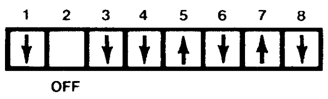


Figure 1-3. Series-IV Development System (Rear View) 121753-3

Booting From the Integrated 5-1/4″ Winchester

To boot from the integrated 5-1/4″ Winchester:

1. Verify the configuration switches are set as shown below.



2. Power up the Series IV by pressing on the breaker switch located at the left rear of the terminal chassis.
3. Verify the system passed the power up diagnostics test. If it did not, contact your Intel Service Representative.
4. Turn on any peripherals such as a Winchester disk drive or a printer. When using 740 Hard Disk Drives, press the STOP/START button and wait for the READY light before proceeding.
5. Press the RESET switch (see figure 1-3) to boot the operating system.
6. Verify the diagnostics test passed. If it did not, refer to the *Intellec Series-IV Installation and Checkout Manual*, 121757.
7. Enter the appropriate date and time as requested. The format is MM/DD/YY for the date and HH:MM:SS for the time.
8. Enter your assigned user name, then your password.

   The first time you log on after installing your system, enter the name SUPERUSER. Your associated password is PASSME. This is your system identification until you establish other users on a shared file structure. For information on making your files and workstation accessible to others, see "File Access and Ownership" in Chapter 3 and the USERDEF and CHPASS commands in Chapter 4.

The system is now ready to accept commands. Use the facilities detailed in the *Series-IV Overview*, 121752, to assist you in learning to interface with your system as you design and develop your projects using the Series IV.

Terminating Operation

Terminate User Session

To terminate the user session and to allow another user to start a session:

1. Log off by entering the command LOGOFF or by pressing the appropriate soft key.
2. The new user may now log on by entering his user name and, if appropriate, his password.

The system is now ready to accept commands from the new user.

Power Down Development System

To shut off the development system:

1. Log off by entering the command LOGOFF or by pressing the appropriate soft key.
2. Turn off any peripheral devices.
3. If you are using a flexible disk(s), wait for the drive indicator light to go off (see figure 1-2). Release the drive door latch(es) and remove the flexible disk(s).
4. Move the rocker switch located at the left rear of the terminal chassis to the OFF position.

Types of Disk Files

A disk is either a system disk or non-system disk, depending on its iNDX files.

- A system disk contains the files necessary to boot the operating system.
- A non-system disk contains only the files necessary for the creation and storage of files, leaving more space for data than on a system disk.

When the system is reset with an iNDX system disk in the appropriate drive, the operating system initializes and takes control of the system.

At system start-up, only the essential iNDX files are loaded into memory. iNDX command files remain on disk until you enter a command that calls them. The required program is then loaded into memory and executed. After the command program has completed its functions, the memory it was using is again available. This allows efficient use of the operating system and lets you reserve most of the memory space for your work.

Each disk file has a name. iNDX program files come with assigned names. You must assign a name to each file you create. To access a file, you need only specify its name, not its address.

The basic types of files are:

- iNDX system files containing both the basic system programs and the command programs.
- User created files (data and program).
- Directory files.

Disk Formatting Procedures

You must format a blank disk before using it. The following examples provide step-by-step disk formatting instructions for flexible disks, the integrated 5-1/4″ Winchester disk and the 8″ Winchester disk.

Example 1 is used for formatting disks on a system with two flexible disk drives. Examples 2 and 4 are used to format disks on a system with a single flexible disk drive and an integrated 5-1/4″ Winchester hard disk. Example 3 is used to format the 8″ Winchester hard disk.

CAUTION

The FORMAT command deletes any files that were on a previously formatted disk.

Example 1 - Formatting a Flexible Disk in a System Containing Two Flexible Disk Drives

1. Log on to your Series IV.
2. Insert the disk to be formatted into drive 1.
3. Enter the following command:

```
FORMAT FL1 VOLUME NAME
```

Example 2 - Formatting the Integrated 5-1/4″ Winchester Disk

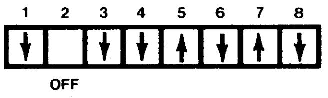
1. Log on to your Series IV.
2. Format the integrated 5-1/4″ Winchester with the following command:

```
FORMAT WMO MINI-WINCHESTER VOLUME NAME
```

3. Copy the iNDX files from the flexible disk to the integrated 5-1/4″ Winchester with the following command:

```
COPY DISKETTE VOLUME NAME TO MINI-WINCHESTER VOLUME NAME
```

4. Log off your Series IV.
5. Reset the configuration switches to boot off the integrated 5-1/4″ Winchester.



At this time, you may also want to copy the iNDX CUSPS files to the integrated 5-1/4″ Winchester. Log on to your Series IV and insert the iNDX.CUSPS diskette. Use the command in step 3 to copy from the flexible disk to the integrated 5-1/4″ Winchester.

Example 3 - Formatting an 8″ Winchester Disk

1. Log on to your Series IV.
2. Format the Winchester with the following command:

SUBMIT */operating system ID/*SYSTEM.BUILD (FLO, *p1*, *p2*)

where

operating system ID	is iNDX.S31 for a Series IV/3 and iNDX.S41 for a Series IV/4.
FLO	is the volume root directory name of the system flexible disk.
p1	is the volume root directory name of the target system device.
p2	is the fixed device name of the target system.

This command will also copy the system files of *operating system ID Vx.y* to the Winchester.

To build the system on the disk, name the volume of the Winchester (e.g., WINI0). By naming the Winchester, you create the volume; by running SYSTEM.BUILD, you set up the necessary directories.

For example, to format the Winchester (WD0), and to copy the system files from the *operating system ID Vx.y* (iNDX.S31) to the Winchester (WINI0), type:

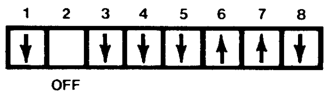
no space

```
SUBMIT /iNDX.S31/SYSTEM.BUILD (FLO, WINI0, WD0)
```

This formatting and copying process takes approximately one half hour. Once you enter the SUBMIT command, you are not required to interact with the system until the SUBMIT process is completed.

3. When the copy is complete, log off your Series IV.

4. Reset the configuration switches to boot off the Winchester.



At this time, you may also want to copy the iNDX CUSPS files to the integrated 5-1/4″ Winchester. Log on to your Series IV and insert the iNDX.CUSPS diskette. Use the command SUBMIT /WINI0/CUSPS.COPY (WINI0) to copy the CUSPS files to the integrated 5-1/4″ Winchester.

Example 4 - Formatting a Flexible Disk in a System that Boots From an Integrated 5-1/4″ Winchester Disk

1. Log on to your Series IV.
2. Insert the disk to be formatted into drive 0.
3. Enter the following command:

```
FORMAT FL0 VOLUME NAME
```

CHAPTER 2
HUMAN INTERFACE

The human interface module of the Series IV provides a command language interpreter and a local command invocation/termination facility. The command language interpreter (CLI) processes command input, initiates command execution and, when execution is complete, prompts for another command. A batch command processing facility is available through the SUBMIT, BACKGROUND, and EXPORT commands. CLI also provides line editing facilities.

The iNDX operating system will ask you to log on. When log on is completed, the operating system is in command mode (the prompt (>) is displayed). At this time, you may enter any of the valid command sequences. To terminate the session you must log off at the terminal. If you want to terminate execution of a file or command, you must use the BREAK key on the terminal keyboard.

The Intellec Terminal

This section describes the display area and keyboard of the Intellec terminal.

Display Screen

The display screen (shown in figure 2-1) is partitioned into four display fields. Information entered in a given field never carries over into another field.

Scrolling Field:	Lines 1-23 form the Scrolling Field. Each line is 80 characters wide. User-entered text will appear in the Scrolling Field. If more than 23 lines are entered, the field scrolls up one line. The topmost line disappears and the additional information being entered appears on Line 23.
Message Field:	The Message Field occupies character positions 1-60 (left-justified) of line 24. Many system programs use the Message Field as a display area to avoid disturbing text that appears in the Scrolling Field.
Operating System Field:	Character positions 61-80 of line 24 form the Operating System Field. The operating system uses this field to display Job Control messages, fatal error messages, the names of mounted volumes, and the names of background and remote jobs.
Prompt Field:	The Prompt Field occupies line 25. This field is used by the operating system to display Menu entries. A maximum of 8 entries may appear on line 25. Each of the entries is associated with one of the 8 Function (or Soft) keys. Reverse video and displaying the prompt field as two side-by-side subfields of four entries each shows the association between the Prompt Field and the Soft keys.

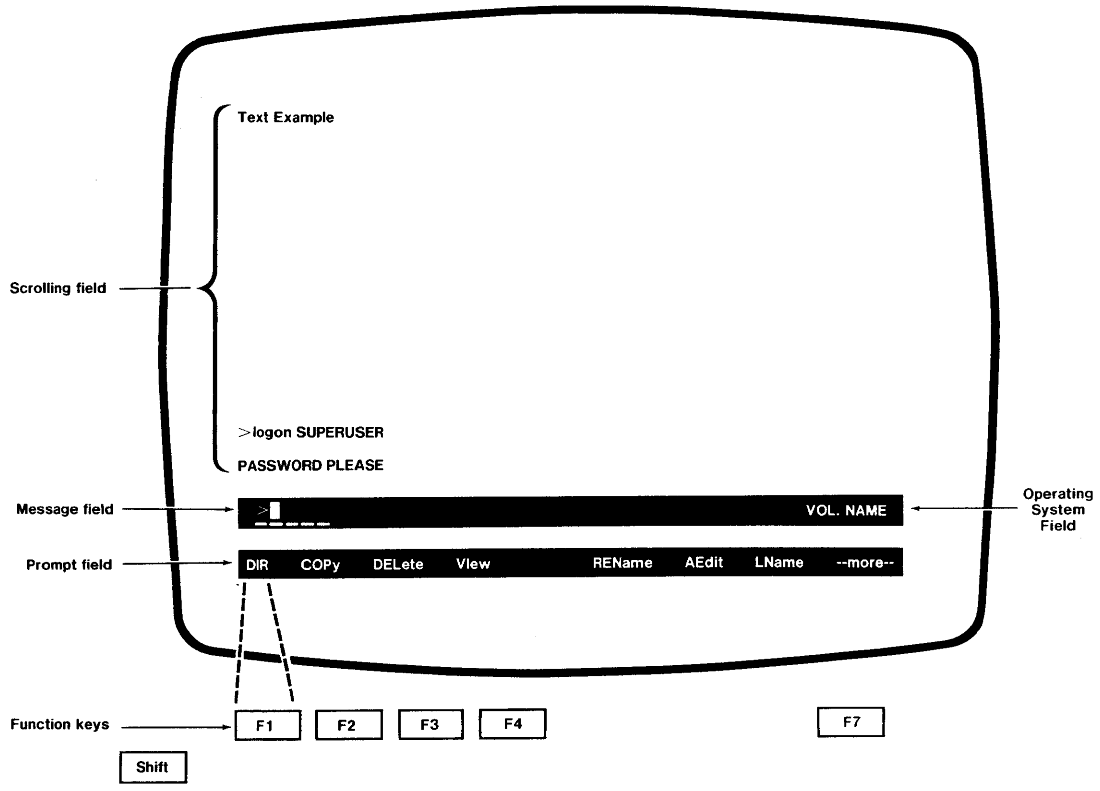


Figure 2-1. The Display Screen 121753-4

Character Display

Characters in the display fields will appear on the screen overlined, in reverse video, blinking or highlighted.

Cursor:	The cursor appears on the screen as a nonblinking, reverse video rectangle. The cursor moves 1 character position to the right with each keystroke until it reaches Column 80 (the right-most column). Pressing the Return key causes the cursor to move to the initial (left-most) character position of the next line. If the cursor is on the bottom-most line of text (i.e., Line 23 of the Scrolling Field), the text is scrolled up one line.

The Keyboard

The keyboard, shown in figure 2-2, is your interface with the system. From the keyboard you control the system, enter data and commands, and request data.

The data you enter at the keyboard is stored in a line editing buffer until you press the RETURN key.

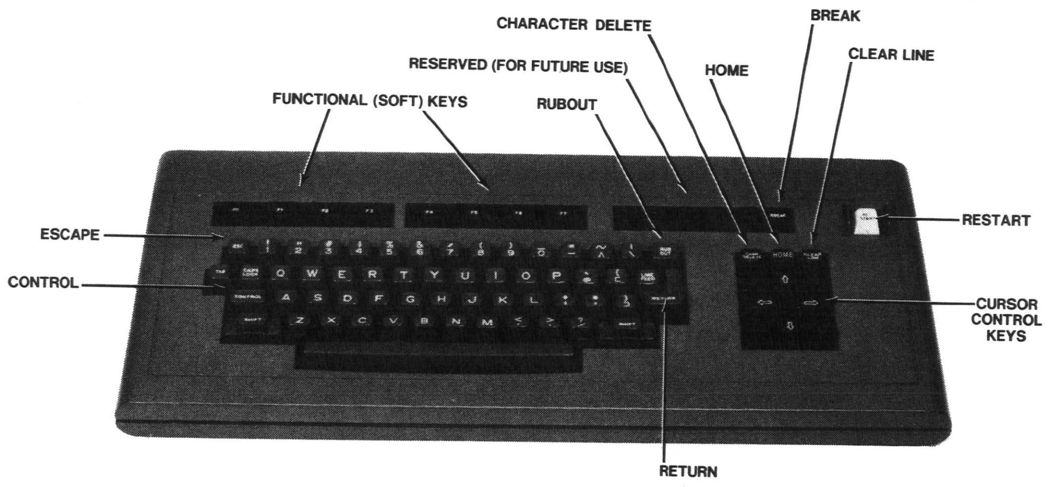


Figure 2-2. Series-IV Keyboard

121753-5

Key Clusters

The keyboard is organized into four clusters of keys: alphanumeric keys, function keys, reserved keys, and edit keys. Following is a description of each key cluster.

Alphanumeric Keys:	Each of these keys generates the character inscribed on the keycap. The Shift key or the Caps Lock key is used to produce the uppercase character.
Shift Key:	If two characters are inscribed on a given alphanumeric key, the Shift key must be pressed with the alphanumeric key to produce the character inscribed on the upper part of the keycap. When the Shift key is not used, the lower character is generated. For alphabetic keys, using the Shift key produces the uppercase letter. Holding down the Shift key and then pressing a Function key (see below) produces the Help Text for the command associated with the given Function key.
Caps Lock Key:	The Caps Lock key functions only with the 26 alphabetic keys. It allows you to enter a series of uppercase alphabetic letters without having to hold down the Shift key. To enter a string of uppercase characters, press the Caps Lock key to its lower, "locked-in" position. To generate lowercase characters again, press the Caps Lock key again, returning it to its upper "unlocked" position.
Function Keys:	When the Command Line Interpreter is executing, these keys are used in conjunction with the Menu items in the Prompt Field (line 25 of the Display Screen). Each of the eight Function keys is associated, position-by-position, with one of the Menu items. To select a given Menu item, press the key associated with that item. For example, to select the third Menu item (counting from left-to-right), press the key inscribed F2 (F0, F1, F2 being the first three keys from left-to-right). Pressing a Function key while holding down the Shift key produces the Help Text for the menu entry associated with that key.

Reserved Keys:	The cluster of five keys located in the upper right of the keyboard are reserved for future system use.
Edit Keys:	The Edit keys consist of the keypad on the right of the keyboard and several other special keys along the right and left edges of the alphanumeric key cluster. Of the eleven keys in the right keypad cluster, only seven have inscribed keycaps. The remaining keys do not have assigned functions. The line editing keys are described in table 2-1.

The BREAK and RESTART keys are special keys also. The BREAK key, located to the right of the reserved key cluster, aborts the execution of a job and returns the system to the interactive level (foreground). CONTROL-C has the same function if the job is executing in the foreground mode.

The RESTART key is reserved for use by field service personnel so they can check the 8086/8088 side of the monitor.

Table 2-1. Line Editor Features

Key Name	Function
RETURN	1. Terminates the line at the current cursor position. 2. Enters the command line into the system.
ESCAPE (ESC)	1. When entered as the first character in a command line, it recalls the last line to the display. 2. Terminates the line at the right margin, not at the current cursor position (as with RETURN).
RUBOUT	Deletes the character to the left of the cursor and moves the cursor left one position.
CTRL X (CONTROL plus X)	Deletes all characters in the current line which are to the left of the cursor. The remainder of the line is re-displayed (left-justified) with the cursor at the left margin of the line.
CTRL A (CONTROL plus A)	Deletes all characters from the current cursor position to the end-of-line. The cursor position does not change.
DEL CHAR	Deletes the character at the cursor location. The cursor position does not change.
CLEAR LINE	Deletes the entire line and returns the cursor to the start position for that line. Control remains in the line editor.
↑ (up arrow)	Moves the cursor up one line; retains column positioning.
↓ (down arrow)	Moves the cursor down one line; retains column positioning.
→ (right arrow)	Moves the cursor one position to the right but not past the current end-of-line.
← (left arrow)	Moves the cursor one position to the left but not past the starting position.
HOME	Moves the cursor position to the current end-of-line. If the last character entered was a left arrow, this key moves the cursor to the starting position.
CTRL S (CONTROL plus S)	Stops output to the console.
CTRL Q (CONTROL plus Q)	Resumes output to the console.

Console Operation

The iNDX operating system provides two features that help you enter commands: the Syntax Guide and the Help Text. The Syntax Guide presents each command and all of its options to you as menu entries. When you first begin using the system, open this manual to Chapter 4 as you enter the commands. With experience, however, you will be able to enter the commands by using only the information presented to you on the screen; you probably will not have to remember them or look them up in the manual.

The second feature of the iNDX operating system is the Help Text, an English language description of a given command, option, or executable file. If you are unsure of the function or operation of any command or any of its elements, you can obtain the appropriate Help Text simply by pressing the Function key (F0-F7) associated with that element while holding down the Shift key.

Entering Commands—Menu Selection

Interactive Mode

The Syntax Guide displays commands, options, and executable files as Menu entries in the Prompt Field (line 25 of the display screen). Up to eight entries—grouped into two blocks of four entries each—can be presented at one time. This arrangement emphasizes the correspondence between the menu entries and the 8 Function keys that can be used to select them. To select a given menu entry, either enter it from the keyboard or press the Function key spatially associated with it. The first (left-most) menu entry is associated with the F0 key, the second with the F1 key, and so on, to the eighth (right-most) entry, which is associated with the F7 key. Usually, the eighth menu entry is "--more--," indicating that additonal menu entries can be displayed by pressing the F7 key. The menu scrolls, eventually returning to the initial display. If you use the keyboard entry method rather than the function key method, you need to know if the Fill option is operative (see the description of the Fill command in Chapter 4). If Fill is operative, enter only the letters shown in uppercase in the menu line because the system fills in the remainder of the command automatically. If Fill is inoperative, you must enter the complete command.

For each menu entry, the accompanying Help Text can be obtained by holding down the Shift key while pressing the appropriate Function key. For example, consider the following 8 menu entries:

```
DIR   COPy   DELete   VIew     REName   AEdit   LName   --more--
```

To select the DELETE command, you could press the F2 key or, assuming for the example that FILL is operative, you could enter the characters DEL. (If FILL was inoperative, you would have entered DELETE). If you need more information about the Delete command, press the SHIFT and F2 keys simultaneously to obtain Help for the Delete command.

Once you have selected Delete from the menu line (either by pressing the Function key or by entering the command from the keyboard), the menu line will no longer display the original eight entries. Instead, the menu line will now contain the following message:

ENTER *file name*

where *file name* is the file you wish to delete. Suppose you do not clearly understand what is required as an entry. Since the original menu line is not on the screen, you cannot obtain Help by pressing Shift-Function keys. The Syntax Guide allows you to "back up," thus recalling the previous menu line. To back up, press the cursor left (←) key. The original menu line will now reappear.

To obtain more information about the Delete command, press the Function key associated with the Delete menu entry while holding down the Shift key.

After reading the Help Text, you can return to the request for a file name by pressing the cursor right (→) key. As long as you enter valid characters the menu line will remain the same. When you have completed the file name, enter a delimiter by pressing the space bar. Note that the menu line has changed again and is now listing the options for the Delete command, as shown below:

```
Dir   Query   --exec--
```

You can select either (or both) of the options by pressing the Function key associated with them (i.e., F0 and F1), or by entering D or Q at the keyboard (Dir or Query if Fill is inoperative). If you want more information about the two options, press both the Shift key and Function keys associated with the desired option. If you have already selected an option but you decide you need information about it, "back up" the menu by using the cursor left key.

After you have entered the entire command line, execute it by pressing the Function key associated with the "--exec--" entry or by pressing the Return key or Escape keys.

Command Language Interpreter

The Command Line Interpreter (CLI) is a program with which you directly interact. The CLI prompts you for input with the > prompt, accepts a complete command from the input device, substitutes the CLI variables with their values, and then loads and invokes the requested program. A complete command may require several input lines, each with a maximum of 128 characters. Each input line except the last contains the continuation character, an ampersand. When the CLI detects the ampersand, it issues a continuation prompt (> >) and allows you to enter another physical line. Only when a complete command line has been entered is the program loaded and invoked.

If you want to include comments in a command line, use a semicolon (;). All characters between the semicolon and the return are recognized as a command comment and are ignored in execution.

Command Delimiters

A command consists of a sequence of characters. When a command is examined by the system prior to execution, special characters called delimiters are used to divide the command into words that are treated as units. The following characters function as delimiters:

!	<
#	=
$	>
%	[
&	\
(	]
)	.
+	|
,	space
-	
;	

A delimeter can be used within a string by enclosing the string within quotation marks.

Pathnames must not be broken by the ampersand. If you enter a pathname as part of a command and the pathname contains a delimiter, you must enclose the pathname in quotation marks so the system will treat it as a unit. For example, if you enter the pathname /VOL1.A/A!B as part of a command, you must enclose it within quotes:

```
COPY  "/VOL1.A/A!B" TO /VOL2.B
```

Command Line Input

Command input lines are entered one at a time using the system line editor. After the Series IV has been initialized, you can enter commands at the terminal whenever the iNDX prompt (>) appears. A single input line may consist of up to 80 characters. For each input line, the cursor will appear in character position 3, directly following the prompt character. Enter your commands and use the edit control characters described in table 2-1.

The terminal supports "type ahead." This means that up to 32 keystrokes are saved in the buffer when the system is not ready for them. Once 32 characters are saved, you will hear a beep; any excess characters will be discarded.

The command line is not complete until a RETURN or ESC key is used to terminate the line. During line input, the cursor (a video-enhanced block) indicates the position of the next character. In line edit, the cursor does not move off the current line or to the left of the starting position (the third character position). If you enter more characters than can be displayed on the current screen line, an exclamation mark (!) will appear in position 79 and the cursor will appear in position 80.

CLI Variables. CLI variables are symbols that have string values associated with them. The command language allows these variables to be defined and referenced within a command file. The scope of CLI variables is restricted to the command file in which the CLI variables are defined. The name of a CLI variable can be a maximum of six alphanumeric characters long. The first character of the name must be a letter. Letter case is not significant. The value associated with a CLI variable is a string with a maximum of 508 ASCII characters. A reference to a CLI variable consists of a percent sign (%) followed by the name of the CLI variable.

The CLI must be able to distinguish the variable reference from the surrounding text. You can guarantee this by:

1. Following the variable name with a delimiter character.
2. Enclosing the variable reference in matching quotes. The quotes are removed if the program obtains its arguments through the DQ$GET$ARGUMENT primitive.
3. Making the variable name a full six characters long. The CLI stops looking if a delimiter has not been encountered within six characters.

Whenever a reference to a CLI variable is encountered in a command line read by CLI, the interpreter replaces the reference with the current value of the CLI variable before executing the command. If the referenced CLI variable is undefined, the reference becomes the null string. Examples of CLI variable use appear in figure 2-3.

There are two different types of CLI variables: system-defined CLI variables and user-defined CLI variables. No more than ten CLI variables can be defined within one command file, including both system-defined and user-defined variables.

System-Defined CLI Variables. One system-defined CLI variable is provided in the Series IV: 'STATUS'. Any reference to it takes the form '%STATUS'. At any given point, the value of STATUS represents the completion code returned by the last DQ$EXIT call executed (see Chapter 7). The completion code is converted to a string of ASCII decimal digits, thereby expressing the value of the completion code in decimal notation (leading zeros are suppressed). Control structures, together with the STATUS variable, make possible the conditional execution of subsequent steps in a command file.

User-Defined CLI Variables. You can also create CLI variables. No separate command exists for defining CLI variables. Use the SET and READ commands (see Chapter 4) to create your variable.

Commands for Manipulating CLI Variables. The SET command is an assignment statement for CLI variables. If the receiving variable does not exist, SET creates it. The string concatenation of any combination of literal strings and the values of CLI variables can be assigned to a CLI variable (either system-defined or user-defined), as shown in the following examples.

Assuming FILE and EXIT are user-defined CLI variables with the following values:

%FILE = 'aprog'
%EXT = 'obj'

The statement

```
SET NAME TO '%FILE.%EXT'
```

results in

%NAME = 'aprog.obj'

In this example, the quotes surrounding the values of the CLI variables are not part of the values, but are included to separate the string values from the surrounding text.

As CLI processes a command line, the following sequence occurs: (1) the line is read (2) the line is scanned for references to CLI variables and all substitutions are performed (3) the line is parsed as a command. The literal quotes around the object following the TO are necessary because the values of the CLI variables being substituted may contain delimiters.

Examples of CLI Variable Substitution. Assume the following CLI variables have been defined:

%FILE = 'mod'
%NUMBER = '1'
%EXT = 'p86'

Consider the following command lines before substitution after substitution, and at execution (i.e., the command tail as it appears when retrieved by the DQ$GET$ARGUMENT UDI primitive).

before:	plm86 %FILE%NUMBER.%EXT
subst:	plm86 mod1.p86
exec:	plm86 mod1.p86

```
before:   plm86 "%FILE"A%NUMBER.%EXT
subst:    plm86 "mod"A1.p86
exec:     plm86 modA1.p86
```

The quotes are necessary because the CLI would interpret %FILEA%NUMBER as a command line containing a reference to a CLI variable called FILEA.

```
before:   plm86 %FILE%NUMBERA.%EXT
subst:    plm86 mod1A.p86
exec:     plm86 mod1A.p86
```

Quotes are not necessary in this example because the CLI knows that variable names are a maximum of 6 characters long. CLI assumes the second variable reference is to NUMBER rather than to NUMBERA.

If a literal percent sign needs to be included in a command line it can be surrounded by matching quotes:

```
before:   a'%'FILE.obj
subst:    a'%'FILE.obj
exec:     a%FILE.obj
```

Command Files

Commands are normally read from the physical console. However, you can create permanent files that contain lists of commands. These permanent files are called command files. You can execute command files in the foreground of your workstation by using the SUBMIT command (see Chapter 4); in the background of your workstation by using the BACKGROUND command (see Chapter 4); or at a remote workstation by using the EXPORT command (see Chapter 5).

Refer to figure 2-3 for an example of command file usage. In addition to the normal console commands, the Series IV has control commands that provide, at run time, conditional or repetitive execution of a set of commands within a command file. CLI variables may also be defined within the command file.

Dynamic File Creation

When you enter a command to CLI from the keyboard, the syntax builder is invoked. Prompts are always displayed, although you can disable command keyword completion (refer to the FILL command in Chapter 4). If you attempt to create a command file with a standard text editor, the syntax builder prompts will not occur. If you were dependent on these, you might have to check a manual or the corresponding reference card to create a command file. To avoid this inconvenience, CLI provides the BATCH command. The BATCH commands allow you to use the syntax builder to create, write, and execute a command file. Another advantage of the BATCH command is that since it is part of the CLI, less memory is required to invoke it than is required to invoke a separate text editor. When the BATCH command is invoked, only the keyword BATCH and the pathname of the command file are specified.

If the command file already exists, the syntax builder's editor can be used to alter the contents of the command file. If the file does not exist, it is created and then written at the end of the edit process. When the new command file is complete, the BATCH command prompts you: abort, write it without executing it, execute it in the foreground or background, or export it to a remote station for execution. If you want to execute it, BATCH allows you to write and execute the file, or allows you to execute it only (a temporary file is created and then deleted after execution). If you

```
Command File (compil.csd)

    OPEN FILES.NRM              ; get parameter file
    SET LINKEM TO ' '               ; initialize
                                    ; string to
                                    ; hold link file
                                    ; name

    REPEAT
        READ NAME                   ; get module
                                    ; name
        WHILE %NAME <>' '           ; exit loop at
                                    ; end of
                                    ; parameter file
        ;
        ;   compile one module
        ;
        PLM86 %NAME.P86 DEBUG COMPACT
        ;
        ;   build string of file names for link step
        ;
        IF %LINKEM = ' '
            SET LINKEM TO '%NAME.OBJ'
        ELSE
            SET LINKEM TO '%LINKEM,%NAME.OBJ'
        END
    END
    ;
    LINK86 %LINKEM,:F0:SYSTEM.LIB TO DRIVER BIND

Parameter File (files.nrm)

    driver, mod1, mod2

Invocation Line

    SUBMIT compil
```

Figure 2-3. Command File Example

execute a file containing references to formal parameters, BATCH prompts you for the actual parameter values to be used. Refer to Chapter 4 for a detailed description of the syntax and operation of the BATCH command.

Log Files

The Human Interface provides a log facility that lets you specify that output written to the logical console output device should also be written to a mass-storage file. In a foreground job, normal console output is displayed on the physical screen. If a log file is active, the same output is also written to the log file. In a background job, the normal console output goes to the byte bucket. A log file can be requested in either of two ways: by selecting the LOG command or by specifying the LOG option of the SUBMIT, BACKGROUND, and EXPORT commands. (The LOG command is valid only from the keyboard. See Chapter 4 for a description of the LOG command.)

The log file is the only attribute of a command file environment which is inherited by nested command files. If a log file is active when a SUBMIT command is issued, the console output from the newly submitted command file is also written to the currently active log file unless the LOG keyword is specified on the SUBMIT command. If a log file is active when a SUBMIT command with the LOG keyword is issued, the current log file is detached before the new log file is created. When the inner nested command file has finished executing, its log file is detached, the log file of the outer command environment is re-attached, and the file pointer is positioned at the end-of-file. The log file of the outer command environment (either the keyboard or another command file) resumes with the next command after the SUBMIT (with LOG keyword).

Parameter Substitution

Actual parameters can be specified when a command file is submitted for execution. The command file can have formal parameters of the form %*n*, where *n* is a decimal digit (0-9). At submit time, a list of actual parameters is supplied along with the name of the command file to be executed. Before the command file is executed, the CLI creates a new copy of the command file where all occurrences of formal parameters have been replaced by the corresponding actual parameters. The formal parameter %*n* is replaced by the (*n* + 1)st element of the list of actual parameters (%0 is replaced by the first list element, etc.). The parameter replacement is done by scanning each line in the command file once from left-to-right. Every occurrence of the string %*n* is replaced by the corresponding actual parameter in a string substitution. If the actual parameter is enclosed in quotes, the quotes are removed before the substitution is performed. If a formal parameter has no corresponding actual parameter, the replacement is performed using the null string.

Parameter Files

To increase the power of command files as utility tools, the command language has commands that allow the values of CLI variables to be read from mass-storage files. These files, referred to as parameter files, are manipulated using the OPEN and READ commands. At most, only one parameter file can be open at any given time. The OPEN statement allows any pathname to be specified as a parameter file. The READ command treats the parameter file as a byte stream subdivided into strings by delimiters, and specifies a list of CLI variable names. The READ command proceeds from the current file pointer position in the parameter file and assigns a string to each variable in the list. If end-of-file is detected on the parameter file during a READ, the parameter file is closed. If the list contains more variables than the number of strings in the file, the variables without corresponding strings are set to the null string.

Suppose file "files.nrm" contains the following two logical lines:

```
driver, mod1
mod2, mod3
```

The execution of the command file fragment

```
OPEN FILES.NRM
READ FILE1,FILE2,FILE3
```

results in values of the CLI variables FILE1, FILE2, and FILE3:

```
%FILE1 = 'driver'
%FILE2 = 'mod1'
%FILE3 = 'mod2'
```

The quotes are not part of the values of the variables. If a subsequent READ command is issued, the string retrieved is 'mod3'. Thus, general purpose utility command files that can process groups of related modules, whose names are specified by parameter files, can be constructed. An example for such a command file is given in figure 2-3.

System-Designated Device Names

The following device names are defined by the operating system:

:TI:	Serial channel #1 input
:TO:	Serial channel #1 output
:LP:	Line printer (local)
:SP:	Spool printer
:CI:	Console input (typically Series-IV keyboard in foreground)
:CO:	Console output (typically Series-IV display in foreground)
:BB:	Byte bucket Though nonexistent, the byte bucket is treated as a real device by the commands. The byte bucket receives data you want to discard. Writing to :BB:, always successful, simply discards data. Reading from :BB: returns an end-of-file (i.e., a zero byte read).
FL0, FL1	Flexible disks
WM0	Integrated 5-1/4" Winchester disk
WD0, WD1	Winchester 35 MB disk
HD0 thru HD3	HD5440 hard disks

Job Control

A job consists of a sequence of commands that perform activities. A job is explicitly created by the LOGON command and terminated by the LOGOFF command, or implicitly created by a BACKGROUND command. The type of job determines the run-time environment created. The run-time environment consists of the standard input and output files, a set of logical names, the STATUS variable, CLI variables, and the user name and any associated access rights. Jobs are independent of each other; changes made in one job do not affect another job.

Foreground Job

The only job you interact with is a foreground job. It is created when you log on to the system using the LOGON command. It is terminated when you log off. The physical console is the standard input and output device.

Background Job

A background job, created by the BACKGROUND command, allows you to run a batch job in the background concurrently with the foreground job. Syntactically, the BACKGROUND command is similar to the SUBMIT command: you specify the name of a file that contains a sequence of commands to be executed. Upon receiving the BACKGROUND command, job control creates a background job environment. It then implicitly logs you on and assigns to a log file the input console to the disk file that contains the commands and the standard output device (unless you have requested that no log file be used, in which case the standard output device is the byte bucket). The command line interpreter is loaded and the execution of the background job starts. When the command file is exhausted, job control logs you off and deletes the background job. Once created, a background job has no relationship to the foreground job from which it was created; nor does it inherit any environmental information (e.g., logical names or CLI variables) from the foreground job.

The background job has no physical console attached to it. Thus, certain programs cannot be executed in the background and certain primitives are disallowed. A screen-oriented editor or debugger should not be invoked from the background. If a program running in the background calls DQ$TRAP$CC (see Chapter 7), the call returns the message EXCEPT = E$OK, but no action is taken. If CONTROL-C is typed, the CONTROL-C handler of the foreground job is always invoked. A background job ignores a CONTROL-C request.

CHAPTER 3
FILE SYSTEM MANAGEMENT

The iNDX distributed file system offers a hierarchical file structure that provides:

- multiple user access to shared data and directory files
- owner controlled access (World and Owner to the files on mass storage devices)
- a list of files that reside in the directories
- the ability to create new directory and data files while other users are accessing shared files
- flexibility in file maintenance
- an archiving facility for files stored on the shared disk

In previous versions of file systems, files and collections of files were tied to the media (disk) and the disk drive (physical device) where the files were stored. Thus, the terms disk, directory, directory identifier, logical device name, physical device name, and disk drive were functionally identical and could be used interchangeably.

The additional functionality of the iNDX structure requires redefinition of these terms. In this manual, the terms are used as follows:

- Disk - the media where directories of files can be stored.
- Directory - a logical collection of files stored on a disk.
- Directory File - a file that stores information about a directory.
- Directory Name - a user specified label for a directory (SYSTEM.DSK).
- Directory Identifier - a logical name (*n*) or fully qualified pathname used to access a directory.
- Disk Drive - a machine used to access a directory stored on a disk.
- Physical Device Name - a label assigned to a physical device (line printer, flexible disk drive 1, Winchester disk 1).
- Logical Device Name - a label (:BB:, :CO:, :SP:) assigned to a logical device (byte bucket, console, spooler queue).

iNDX File Structure

The iNDX file system is structured hierarchically. This structure resembles an inverted tree (see figure 3-1).

The root or origin of the file system is called the Logical System Root (in the first tier of figure 3-1). It "connects" the volumes within the file system (shown as the second tier). Each volume corresponds on a one-to-one basis to a physical mass storage device. Thus, in the figure, VOL 1.A could be a flexible disk, VOL 2.B a Winchester disk, and VOL 3.C a hard disk. Each volume is further divided into files (shown as the third tier). Each file may be either a data file or a directory file. Data files contain only data; directory files may contain both references to data files and additional directory files.

Pathnames

The files (data and directory) can be traced down through the file structure by a pathname. The pathname identifies every volume and directory from the logical system root to the data file. The pathname for the file pointed to by the arrow in figure 3-1 is /VOL1.A/DIRB.EXT/FILE3.EXT.

The pathname /VOL1.A/DIRB.EXT/FILE3.EXT is a fully qualified pathname because the slash (/) acts as a delimiter between the names of the volume and the various directories in the hierarchical path.

To directly access FILE3.EXT in figure 3-1, you may assign a directory identifier (X) to the fully qualified pathname of the directory with the LNAME command, as in the following example:

```
LNAME DEFINE X FOR /VOL1.A/DIRB.EXT
```

The pathname for this file is now X/FILE3.EXT.

For details and restrictions of the LNAME command, see Chapter 4.

Wildcard Filenames

The COPY, DELETE, ACCESS, DIR, and CHOWNER commands allow you to specify filenames using a wildcard element to replace one or more characters in the last component of a filename. The wildcard elements can appear only in the last component of the filename.

The two wildcard characters are the asterisk (*), which matches any number of characters in the final path component, and the question mark (?), which matches any single character in the final path component.

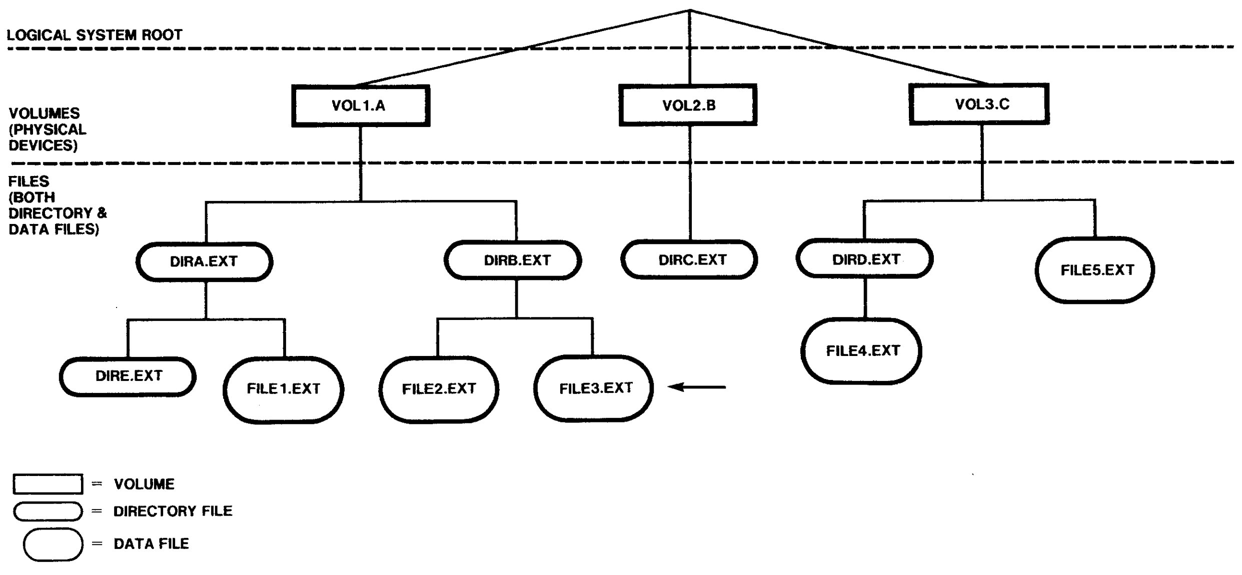


Figure 3.1 Hierarchical File Structure 121753-6

Thus, the wildcard pathname /FAT* will match the files /FATCAT, /FATLADY, /FATCITY, etc. Note that the period (.) is treated like any other character. /FAT* matches /FAT.CITY, /FATC.ITY, etc.

The wildcard pathname /FAT?AT will match the filenames /FATCAT, /FATHAT, /FATBAT, but not /FATXHAT or /FAT.HAT.

More than one asterisk may appear in wildcard filenames. Thus, /*FAT* matches all filenames having the character string FAT appearing between any two other character strings. /AFATCAT and /INFATCITY would be matched.

The other possible combinations of the two wildcard characters such as *B?, ?B* and ?B? are also acceptable.

The wildcard pathname * matches any files in that directory.

Creating Directories

You can create a new directory in the hierarchical tree structure by using the CREATEDIR command described in Chapter 4 . The CREATEDIR command allows you to add new directory files to existing directory files by specifying either the fully qualified pathname (naming all of the branches of the tree) or the directory identifier assigned to the existing directory file.

Directory Maintenance Guidelines

The guidelines that follow will help you take advantage of the versatility and flexibility of the iNDX distributed file system.

To create and maintain your directory and data files:

1. Minimize the number of directories in each volume. To do this, subdivide the directories by project and function.
2. Keep all of the "system files" in one directory file.
3. Create a separate directory of directory files for individual user "miscellaneous files."

NOTE

An extensive structure will retard file accessing and retrieval time. The example figure is for a well defined and heavily structured multiuser environment. Your file system should only be as structured as required by your project. Use this larger structure with a detailed project. Use only two or three levels of file-depth on a small-to-medium size single-user project.

Directory Files

Directory files reside within each volume. Directory files can contain other directory files, or data files. Figure 3-2 shows a volume (WINCH1.VOL) with six directory files: iNDX.SYS, PROJA,DIR, PROJB.DIR, JOHN.DIR, SHEILA.DIR, and LEE.DIR.

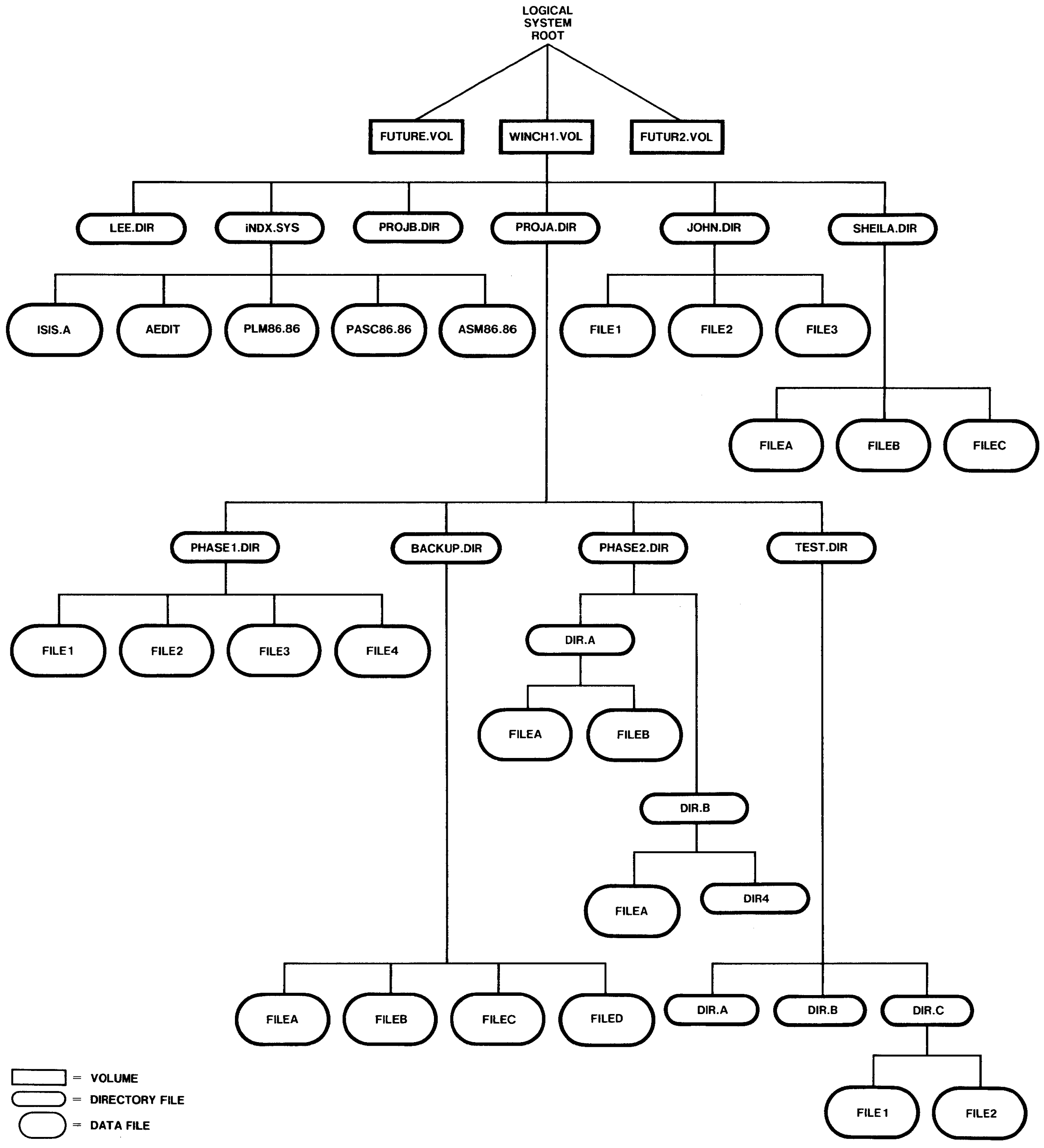


Figure 3-2. Model of iNDX File System 121753-7

The iNDX.SYS directory file contains all the files a programmer would normally look for in a default directory. Those files are:

- ISIS.A (basic system files ISIS.DIR, ISIS.BIN, ISIS.CLI, etc.)
- Command programs (COPY, RENAME, LOGON)
- Text editors (AEDIT™, CREDIT)
- Compilers and Translators (PASC86.86, PLM86.86, ASM86.86, FORT86.86)
- Linkers and Locators (LINK86, LOC86)
- Other tools
- Latest stable version of the prototype software

Each project directory, PROJA.DIR and PROJB.DIR, contains directories that subdivide the project into phases or sections depending on the application. Figure 3-2 shows four directories:

1. PHASE1.DIR—contains various stages of prototype software and all .SRC, .LST, .OBJ and other work files related to this phase of the project.
2. PHASE2.DIR—similar to PHASE1.DIR except this directory is further subdivided for programmer convenience.
3. BACKUP.DIR—stores backup copies of all relevant files related to the project.
4. TEST.DIR—contains separate directories of tests for various aspects of the project. These directories each contain individual test files.

The directory files listed can be controlled as needed by individual programmers.

Each person in the structure has a directory (*name*.DIR) that can be used to store status reports, memos, trip reports and other miscellaneous work.

File Access and Ownership

Every data and directory file in the hierarchical file structure has an "owner." The file is "owned" by the user who created it.

Every file on a shared mass storage device has separate Access Rights for the file owner (Owner Access Rights), the other users (World Access Rights), and the superuser (Superuser Acess Rights). These Access Rights are controlled by access switches that can be manipulated by the file owner from the workstation.

Before accessing a file in a shared file system, you need the appropriate Access Rights to the file. To create a file (directory or data), you must have Access Rights to the directory where the file will reside.

Protecting File Access on a Mass Storage Device

If you are accessing files on a flexible disk, you are the superuser. File protection is provided since you may remove the disk and store it somewhere. Thus, no one else can access your files.

For files stored on Winchester or Hard disks, a file protection system is provided to allow or prevent one user from accessing another's files. The person who manages this system is defined as the SUPERUSER and is responsible for providing system management for mass storage devices that the workstation user cannot provide—devices such as creating and deleting users, managing user ID numbers, and assigning passwords.

Your group should choose one person to become the SUPERUSER. The SUPERUSER should establish a user structure that allows the rest of your group to limit access to their files as they see fit.

Once this is done, each user will have to use his user name and password when logging on. Only the system manager can use the SUPERUSER log on sequence.

Creation and Deletion of Users

The SUPERUSER initially creates (and adds) each user by assigning a unique name, a unique user ID number, a home directory, and a password. The user logs on by entering his name and password at the workstation console. These identification methods guarantee that only authorized users will be able to access the file system. The user ID provides a way of tracking ownership of files throughout the system.

When you remove a user's access rights to shared files, the files belonging to that user are not deleted. An expanded directory listing shows an owner name of "NOT FOUND" for files whose owners have been removed from the structure. The SUPERUSER can then use the CHOWNER command to transfer ownership to another user or can define another user with the same user ID to become owner of the files.

The creation and deletion of users is accomplished via the USERDEF command (see Chapter 4). The USERDEF command recognizes only the predefined user name SUPERUSER. Only with this name can all the SUPERUSER functions be executed. The user name SUPERUSER cannot be deleted from the system. Secondary superusers can be created with distinctive user names and user ID numbers in the range 3-15. (The SUPERUSER has a predefined user ID of 2.) The secondary superusers have all the superuser capabilities except the ability to execute the USERDEF and USERS commands. Secondary superusers may be removed from the structure by the primary superuser.

Management of User IDs

The primary superuser assigns user names and user ID numbers with the USERDEF command. The system keeps a record of these identification assignments for the SUPERUSER. He can access that record via the USERS command (see Chapter 4).

The USERS command can only be used by the primary superuser. The command displays a list of both user names and their associated ID numbers. Valid user ID numbers are in the range 1024-32767.

Assignment of Passwords

The USERDEF command assigns a null password to each newly created user name. The SUPERUSER can then assign passwords using the CHPASS command. Passwords can be unique for each user or can be common to a project or products. The SUPERUSER decides how to assign passwords.

The CHPASS command (see Chapter 4) also changes existing passwords. The SUPERUSER need not specify the old password to cause a change. Passwords are restricted to a maximum of 14 characters.

File System Considerations

Due to the structure of the hierarchical file system, users who have ALL WORLD access rights to a shared file need to be careful when editing it. For example, if you delete part or all of a shared file, no one else can access (that part of) it any longer. Another example is: if user A renames a shared file without informing user B of the new name, B can no longer access that file.

When editing shared files (especially in using the RENAME and DELETE command), document the changes so others can understand how you have altered the files.

CHAPTER 4
COMMANDS

The Series-IV development system provides console commands that enable you to productively interface with the hardware, programs, files, and supporting peripherals. These commands help you to distribute jobs, maintain files, and transport information from other systems to the Series IV.

Command Categories

Except for LOGON, LOGOFF, and TIME, the commands are arranged in four groups:

Control Commands
File Maintenance Commands
Job Management Commands
Media Operation Commands

Control commands help build a command file. You can select or execute commands from the command file to help you in your program.

COUNT	OPEN
EXIT	READ
FILL	REPEAT
IF	SET
LOG	

File maintenance commands are used to establish, change, and maintain files. Use the file maintenance commands when you want to change the file set within your directory.

ACCESS	DISMOUNT
ARCHIVE	LNAME
CHOWNER	MOUNT
CHPASS	RENAME
COPY	SDCOPY
CREATEDIR	SPACE
DELETE	USERDEF
DIR	USERS

Job management commands enable you to distribute your job load efficiently so more than one job can be executed simultaneously.

BACKGROUND	REGION
BATCH	SUBMIT
CANCEL	

Media operation commands can be used to transport information from another medium to the Series IV.

FORMAT	PDSCOPY
FPORT	VERIFY
ICOPY	

Entering and Editing Line Commands

For a full description of how to enter and edit line commands, refer to "Console Operations and Command Language Interpretation" in Chapter 2.

Commands

The remainder of this chapter details the commands of each command category.

Each command—along with its syntax, a description of its functions, and some examples—is listed alphabetically within its respective category.

The LOGON, LOGOFF, and TIME commands—not part of any of the four command categories—are listed first.

LOGON

Syntax

```
LOGON   username   [{INIT (filename)}]
                   [{NOINIT         }]
```

where

username	identifies the user to the operating system.
pathname	is a pathname, wildcard pathname, or both.
INIT and NOINIT	indicate whether the user environment is initialized with a command file (INIT is the default).

Description

This command prevents unauthorized system access on protected device systems.

If you are accessing a mass storage device, the following prompt appears after you enter your user ID:

```
PASSWORD PLEASE
```

Your password, followed by < cr >, must then be entered. Line editing is not allowed when entering the password. Also, the password does not appear on the display screen. If an incorrect character is entered, press < cr >, observe the new prompt, and enter the password correctly.

System access will be granted if the entered *username* is found in the predefined system USERDEF file and the entered password corresponds to the *username.*

The INIT option determines which command file will be executed to initialize the user's execution environment (i.e., the initialization file will normally consist of commands assigning logical names, etc.). If neither INIT nor NOINIT are specified, a default filename of INIT.CSD will be used. *Unless* NOINIT is specified, the following will occur:

1. If a filename is specified, the existence of the file will not be verified. The filename will be Submitted for execution as the initialization file.
2. If no filename is specified, a default filename of INIT.CSD will be used as the initialization file. The existence of this file will not be verified.
3. If a filename entered ends with a period, the filename (with the period truncated) will be used as an initialization file.
4. If the filename entered does not end in a period, the suffix .CSD is appended.
5. In the default INIT case, the operating system looks for the directory INIT.CSD. If the directory is found, it is used as the initialization file.

   If no filename was entered, a search is made to determine if the default file INIT.CSD exists in your home directory. If INIT.CSD does not exist in your home directory, a search will be made to determine whether it exists in the System Volume Root Directory. If the default file INIT.CSD does not exist in either directory, no initialization will be performed.

CAUTION

1. LOGON is implicitly performed for BACKGROUND and [redacted] jobs. In these cases, initialization of the user environment will take place exactly as if you had logged on interactively in Foreground and did not specify either NOINIT or an initialization file (i.e., if an INIT.CSD file exists in your home directory or in the System Volume Root Directory, it will be executed as an initialization file).
2. Since INIT is the default case, you should be aware of what occurs at interactive log on time. If an INIT.CSD file is present on your home directory or on the Volume Root Directory but you do not want that file executed, NOINIT or another initialization filename must be explicitly stated.
3. If an invalid *username* is entered, or an invalid password is entered 3 times in succession, the LOGON process is re-started. Consequently, the INIT/NOINIT filename options must be re-entered.

Examples

1.
```
LOGON HARRY NOINIT
PASSWORD PLEASE: BLUE
```

HARRY logs on to the network and correctly enters the password BLUE. No initialization file is searched for.

2.
```
LOGON HARRY INIT (SET.)
PASSWORD PLEASE: BILE
PASSWORD PLEASE: BLUE
```

This time HARRY logs on and specifies that initialization information is to be obtained from file SET. The password BLUE is incorrectly entered as BILE. The prompt reappears and the password is then correctly entered.

3.
```
LOGON HARRY INIT (ENV)
PASSWORD PLEASE: BLUE
```

HARRY successfully logs on and specifies ENV as the initialization file. Since the filename is not followed by a period, the system searches the user home directory for a file named ENV.CSD to use as an initialization file.

4.
```
LOGON HARRY INIT
PASSWORD PLEASE: BLUE
```

HARRY successfully logs on and specifies the INIT option but does not specify an initialization file. The operating system attempts to execute INIT.CSD as the INIT file.

LOGOFF

Syntax

```
LOGOFF
```

Description

This command terminates your current foreground job. All environment information, including logical names, is deleted.

Examples

1. `LOGOFF`

TIME

Syntax

```
TIME
```

Description

The program will sign on as follows:

```
system id TIME (Vx.y)
DATE: mm/dd/yy TIME: hh:mm:ss
```

where

system id	is the operating system's identification.
V*x.y*	is the operating system's version number.
mm/dd/yy and *hh:mm:ss*	are the current setting of the system clock.

The system then prompts:

```
ENTER COMMAND (SET/EXIT):
```

You can then enter a command in line-edit mode. (All keyboard entry to the TIME command is in full line-edit mode.) TIME recognizes abbreviations of the commands. If the command entered is not valid, TIME will display the following message:

```
INVALID COMMAND
```

and return to the ENTER COMMAND prompt.

If the command entered is EXIT<cr> or EXI<cr> or EX<cr> or E<cr>, the program terminates and control returns to the Command Line Interpreter.

If the command entered is SET<cr> or any of its abbreviations, the following prompt will be issued:

```
ENTER DATE (mm/dd/yy<cr>):
```

The user enters the current date in *mm/dd/yy* format. If *yy* is less than 78, the date is assumed to be in the 21st century. If any of the characters in *mm*, *dd*, or *yy* are not numeric, the following message will be displayed and control will be returned to the ENTER COMMAND prompt:

```
INVALID ENTRY
```

If the entry is valid, TIME will prompt for the current time:

```
ENTER TIME (hh:mm:ss<cr>):
```

Enter the current time in military format (i.e., 1 pm = 13:00:00). If any of the characters in *hh*, *mm*, or *ss* are not numeric, the following message will be displayed and control will be returned to the ENTER COMMAND prompt:

```
INVALID ENTRY
```

If the entry is valid, the system clock will be set to the requested date and time. If the date and time entered do not represent a valid combination, one of the messages below will be displayed and followed by the ENTER COMMAND (SET/EXIT) prompt:

```
EXCEPTION 201AH: INVALID TIME
EXCEPTION 201BH: INVALID DATE
```

After the SET command is executed, the current setting of the system clock is displayed and control returns to the CLI.

Additional Notes

1. When invoked within SUBMIT file processing, TIME will display the current system Date/Time and exit, returning control to the CLI.
2. Only the Superuser may set the system Date/Time. If the TIME cusp is invoked by a NON-SUPERUSER, the current system Date/Time will be displayed and the TIME cusp will return control to the CLI.

NOTE

In a network environment, if the system DATE/TIME is set at a workstation, DATE/TIME will be set at the NRM (via transmission of the request). The NRM will then send Update Time Requests to all of the currently active workstations, thus ensuring that DATE/TIME is synchronized throughout the network.

Control Commands

Following are the control commands.

COUNT

Syntax

```
COUNT n

    [ commands ]

[ { WHILE } argument { =  } argument   ... ]
[ { UNTIL }          { <> }                ]

    commands

END
```

where

argument	is a CLI variable value, a CLI variable name or a parameter.
n	is the number of times the block will repeat.

Description

The Count command allows specified iteration of one or more commands. A decimal number specifies the number of times the loop will be executed. The WHILE, UNTIL, or EXIT options can be used for a premature exit from the loop. More than one WHILE or UNTIL clause may be entered. See the following page to use EXIT.

In the UNTIL option, the command set is executed until the logical comparison evaluates TRUE. The first time the comparison evaluates to TRUE the loop is exited regardless of the value of the counter for the loop. In the WHILE option, the command set is executed as long as the logical comparison evaluates to TRUE. The first time the comparison evaluates to FALSE the loop is exited regardless of the value of the counter for the loop. The logical comparison consists of testing one string against another for equality or inequality. The string may be a CLI variable value, a CLI variable name, or a parameter. Substitutions are made before the command line is executed.

NOTE

The WHILE or REPEAT options may be used as often as necessary in a COUNT block.

EXIT

Syntax

```
EXIT [(argument)]
```

where

argument	is a CLI variable value, a CLI variable name, or one of the ten parameters %0 to %9.

Description

The Exit command allows you to terminate the processing of a command file before it reaches its normal end. Usually, the Exit command will appear in an IF statement that is used to check for proper execution. In that case, the Exit command is used to provide an error escape and a return to the calling command file.

The optional value, if specified, is assigned to the predefined CLI variable name in the calling command file (STATUS). If, for example, you establish the decimal number 5 to reflect malfunction, you can check STATUS in the calling program against a malfunction. If STATUS = 5, you know that the Exit command was executed and a malfunction occurred in the called file. If the optional value is omitted and if the EXIT command is executed in the called program, the value 0 will be assigned to STATUS in the calling program. If a parameter file is open within the current command file, the parameter file is automatically closed and detached. Command files are called (or Submitted) either by other command files or interactively from the keyboard (by the Submit, [redacted], or Background commands).

FILL

Syntax

$$\text{FILL} \begin{Bmatrix} \text{ON} \\ \text{OFF} \end{Bmatrix}$$

Description

The Fill command allows you to enable and disable the Command Completion feature on the Syntax Guide. Remember, each menu entry is presented on the Display Screen in initial uppercase letters and subsequent lowercase letters. When Fill is enabled, you may select a menu entry simply by entering only the uppercase letters. The syntax driver automatically fills in the missing lowercase letters. FILL ON enables the feature; FILL OFF disables it.

The Syntax Guide also has a Noise Word Fill feature. When the Syntax Guide enters options on the Menu Line and only one option is available, the Syntax Guide will automatically enter that word (called a redundant or "noise" word—hence the term Noise Word Fill). When Fill is Off, Noise Word Fill is also inoperative. The ON option is the default case.

IF

Syntax

```
IF    argument  { =  }  argument
                { <> }

      commands

 ┌                                                    ┐
 │  ORIF    argument { =  } argument       [ . . . ]  │
 │                   { <> }                           │
 │          commands                                  │
 │                                                    │
 │ [ELSE    commands]                                 │
 └                                                    ┘
END
```

where

argument	is a CLI variable value, a CLI variable name or a formal parameter.
commands	is a set of one or more console commands.

Description

The IF command provides conditional execution of one command set. The command set choice is based upon the result of successive logical evaluations. In each evaluation, the first argument value is checked for equality or inequality against the second argument value. Each value may be a CLI variable value, variable name, or parameter. (Substitutions for CLI variable names and for formal parameters are made at command file invocation.) Generally, the value of a CLI variable name or parameter represents a filename to be tested to determine if the command set is to be executed on that file. The command set consists of one or more commands.

If the IF comparison evaluates to TRUE, the command set immediately beneath it is executed. If the IF comparison does not evaluate to TRUE, the ORIF comparison is evaluated. If the ORIF comparison evaluates to TRUE, the command set immediately beneath it is executed. If the ORIF comparison does not evaluate to TRUE, the command set immediately beneath the ELSE line is executed. If only the IF comparison is used and it evaluates to FALSE, no commands are executed. The ORIF and ELSE comparison lines are optional. More than one ORIF line may be used. Also, the ORIF comparisons are checked sequentially. Only one ELSE line is allowed, however.

Examples

1.
```
IF %STATUS <> "0"
   READ %ABC
END
```

This command checks to see if the completion code returned by the operating system (which is referenced by the predefined CLI variable name: STATUS) is equal to 0. If the returned completion code equals 0, an open parameter file is read and assigns the value read to the CLI variable whose name is defined by %ABC.

LOG

Syntax

$$\texttt{LOG} \begin{Bmatrix} \texttt{:BB:} \\ \textit{pathname} \end{Bmatrix}$$

where

pathname	is a fully qualified pathname or a logical name.
:BB:	is the Byte Bucket.

Description

The Log command creates a log of all console activity within the file specified. If the output filename specified conflicts with any existing filename, the existing file will be destroyed. The Log continues to function until a second command is issued to disable it. The Byte Bucket (:BB:) is used to disable the log process.

Examples

1. `LOG /ONE.VOL/MARK.DIR/FILES3.EXT/FILES3.LOG`

   A log of console activity is created under the filename ONE.VOL/MARK.DIR/FILES3.EXT/FILES3.LOG.

2. `LOG  :BB:`

   The current log file is closed and further logging is disabled until another Log command is received.

OPEN

Syntax

`OPEN` *pathname*

where

pathname	is a valid pathname.

Description

Before character strings can be read from a file on a mass storage device and assigned to CLI variables, the file must be opened. Only one such file can be open at any one time in a given command file. If a parameter file has already been opened in the current command file, it will automatically be detached and closed by the second Open command before the second file is opened.

READ

Syntax

```
READ variable-name [, . . . ]
```

where

variable-name	is either the system defined CLI variable name (STATUS) or any valid user defined CLI variable name.

Description

After the Open command has been used to access the appropriate file residing on a mass storage device, the Read command is used to take character strings from the file and assign them to the variable names given in the Read command. The Read command begins at the current position in the file and reads a character string for each variable name in the command. The non-alphanumeric characters are used to define where one string ends and another begins.

If the variable names given in the command line outnumber the successive strings available in the file, the variable names for which strings are not present are each assigned a null character string.

If a read is attempted but no file has been opened, an error message will appear on the standard output device and all variable names listed in the Read command will be assigned null strings.

Examples

1.
```
OPEN /SYSTMRT/DOGS
READ NAME1, NAME2, NAME3, NAME4
```

Assume that file /SYSTMRT/DOGS contains only the character string FIDO#ROVER$"OLE-BLUE". The variable name NAME1 will be assigned to the character string FIDO and NAME2 will be assigned the character string ROVER. The character string OLE-BLUE has been placed in quotes so the hyphen will be interpreted not as a delimiter but as the literal hyphen character. Therefore, NAME3 is assigned the character string OLE-BLUE. Since no other character string is available from the file, variable name NAME4 is assigned a null string. If the character string OLE-BLUE did not appear in quotes, the hyphen would have been interpreted as a delimiter, NAME3 would have been assigned the character string OLE, and NAME4 would have been assigned the character string BLUE. Failure to use the quotation marks would thus cause two dogs to be given bad names.

REPEAT

Syntax

```
REPEAT

    [ commands ]

    [ { WHILE } ] argument { =  } argument
    [ { UNTIL } ]          { <> }

    [ commands ]

END
```

where

argument	is a CLI variable value, a CLI variable name, or a parameter.
commands	is a set of one or more commands.

Description

The Repeat command allows one or more commands to be looped. The Repeat command has two optional forms: the WHILE form and the UNTIL form. In the WHILE form, the command set is repeatedly executed (in loop fashion) as long as the logical comparison evaluates to TRUE. The first time the comparison evaluates to FALSE, the loop is exited.

In the UNTIL option, the command set is executed as long as the logical comparison evaluates to FALSE (that is, until it evaluates to TRUE). The first time the comparison evaluates to TRUE, the loop is exited. If neither a WHILE or UNTIL line is used, indefinite command set execution will occur.

The logical comparison involves testing one string for equality or inequality against another string. Each string may be a CLI variable value, a CLI variable name, or a parameter. Substitutions are made before the command line is executed.

NOTE

The WHILE or REPEAT options may be used as often as necessary in a REPEAT block.

SET

Syntax

```
SET variable-name TO ["] value ["]
```

where

variable-name	is a valid CLI variable name (i.e., a string of up to six characters, alphabetic and numeric, the first character of which must always be alphabetic), or the predefined variable name STATUS.
value	is a character string. The quotation marks are necessary only if the string contains non-alphanumeric characters.

Description

The Set command assigns a character string of up to 508 characters to a user-chosen variable name or to the system predefined name (STATUS). The character string may be a simple string or a conglomerate of CLI variable names, CLI variable values, and parameters. In the latter case, the system will first substitute names or parameters with their respective strings, then assign the resulting new name to a pure string. This process is made clearer in the following examples.

Examples

1. Assume the CLI variable names %X, %Y, and %Z have been previously assigned to character strings by the following commands.

   ```
   SET X TO "for"
   SET Y TO "peg thankx"
   SET Z TO "1234"
   ```

   X references the string for, Y references peg thankx and Z references 1234. If you now enter the command

   ```
   SET A TO "%X%Y"
   ```

   the CLI variable name A refers to the string formed by the concatenation of %X and %Y. Thus, A is defined as a reference to the character string "for peg thankx". The optional quotation marks are used to prevent any of the characters of the string from being interpreted as delimiter characters.

2. Now, consider the command SET B TO " ". The CLI variable name B refers to the null string (i.e., a string of zero characters). The null string can be useful when testing job execution. For example, you could use the IF command to test filenames against the null string. As long as the overall routine continues to find files, it will execute the rest of the procedure; once the null string is recognized, an exit from the loop will be taken.

3. `SET TNT TO "%Z"a.c%X`

   In this example, the string "1234" is substituted for the previously defined name, %Z, and the string "for" is substituted for the previously defined name, %X. Between these, the character string "a.c" is entered. The variable name %TNT can now be used to reference the character string "1234a.cfor".

It is also possible to redefine a variable name using the previously defined variable name as part of the new definition. For example, if the variable name %AK47 has previously been assigned to the character string "BANG", then the following command

```
SET %AK47 TO %AK47"--"%AK47
```

reassigns the name %AK47 to be the character string "BANG--BANG."

File Maintenance Commands

Following are the file maintenance commands.

ACCESS

Syntax

```
ACCESS [filename  SET [{USER SPEC  DATA ACCESS SPEC}]]
                                   DIR ACCESS SPEC
```

where

filename	is a pathname, wildcard pathname, or null. Null (entered as a filename) gives a list of the access rights of the directory associated with the null logical name.
SET	declares the specified attributes.
USER SPEC	is null, WORLD, or SUPERUSER.
DATA ACCESS SPEC	is READ, WRITE, DELETE, ALL, or NONE.
DIR ACCESS SPEC	is DELETE, ADD, DISPLAY, ALL or NONE.

Description

The ACCESS function allows you to set and display the user access rights to a file. Any user may use this command to display the owner and WORLD access rights of a file. You may also change both your own and the WORLD access rights. The Superuser can change only his own access rights (unless he is the file owner).

When you set the file access rights, replace the existing access rights with the specified access rights. Previous access rights not explicitly specified in the resetting process will no longer be valid. Following are the valid access rights for data files:

READ
WRITE
DELETE
ALL
NONE

You may specify any combination of READ, WRITE and DELETE. ALL indicates full access whereas NONE indicates loss of all access rights to a file.

Following are the valid access rights for directory files:

DELETE
ADD
DISPLAY
ALL
NONE

You may specify any combination of DELETE, ADD, and DISPLAY. ALL indicates full access whereas NONE indicates loss of all access rights to a directory.

If conflicting access rights are specified, the access rights set are the last specified, as shown in the following example:

ACCESS A SET READ NONE	NONE will be set
ACCESS A SET NONE WRITE	WRITE is set
ACCESS A SET NONE READ DELETE	READ, DELETE are set

If no user is explicitly specified, the access rights of the owner will be changed. When either WORLD or SUPERUSER is specified, only that class of accessor will be altered.

When ACCESS is run in display mode, the following screen display appears:

```
FILENAME   OWNER ACCESS   WORLD ACCESS   SUPERUSER ACCESS
XYZ        REA WRI DEL    REA            REA WRI DEL
```

When ACCESS is run in set access mode, the following message is displayed for each completed operation:

SET ACCESS TO FILE *pathname*

Examples

1. `ACCESS /A SET READ`

   This command replaces the existing owner access rights with access for read only.

2. `ACCESS /A`

   This command lists the access rights of the owner, world, and superuser to file A.

3. `ACCESS /A SET ALL`

   This command replaces the existing access rights with full access rights.

ARCHIVE

Syntax

```
                             ┌M  ⎧{B | S | ON} [MDY [HMS]]⎫┐
                             │   ⎩         T              ⎭│
ARCHIVE old-dir TO new-dir   │C  ⎧{B | S | ON} [MDY [HMS]]⎫│[AND ...]⎧U⎫
                             │   ⎩         T              ⎭│         ⎩Q⎭
                             │OW       owner name           │
                             └F        pathname             ┘
```

where

old-dir	is the name of the directory subtrec to be copied.
new-dir	is the name of the new directory subtree.
owner-name	is the name of a file(s) owner.
pathname	is the file pathname component.
M	is MODIFIED.
C	is CREATED.
OW	is OWNED BY.
F	is FILES.
B	is BEFORE.
S	is SINCE.
ON	is ON.
T	is TODAY.
MDY	is mm/dd/yy.
HMS	is hh:mm:ss.

Description

This command is designed to copy files from one directory subtree to another. The files are stored as they last appeared when the command was executed. When used in the Network mode, ARCHIVE may not represent the state of the files at the start of the operation because files may be modified, created or deleted on the network while ARCHIVE is running.

With the MODIFIED qualifier, ARCHIVE selects only files that have been modified since the last ARCHIVE operation. If the MODIFIED keyword is followed by an optional time qualifier (SINCE/ON/BEFORE/TODAY), ARCHIVE will compare the time qualifier with the time the file was modified; it will then copy those files that have been modified and match the time qualifier. The default time for the MODIFIED option is the time of the last ARCHIVE operation.

With the CREATED qualifier, ARCHIVE compares the date entered with the time the file was created and copies only those files that match the time qualifier. The time qualifier is required for CREATED.

With the OWNEDBY qualifier, ARCHIVE copies files selected on the basis of owner name.

The FILES qualifier allows ARCHIVE to copy files selected on the basis of the pathname component. The qualifier accepts wildcard characters as part of the pathname.

Time Qualifiers

The BEFORE/SINCE time qualifiers allow you to specify a time that is compared with the file create or modified time to qualify a file for archiving. BEFORE allows you to specify files created or modified before the date entered. If a time is not entered, the default time is 00:00:00. SINCE allows you to specify files created or modified since the date entered. If the time is not entered, the default time is 00:00:00.

ON specifies the start of a 24-hour period.

TODAY gets the current date from the operating system and defines a 24-hour period beginning at midnight.

The DATE (required for BEFORE/ON/SINCE) consists of the date in the form mm/dd/yy and, optionally, the time of day in the form hh:mm:ss. The DATE must be entered as two decimal digits. The time should be entered in 24-hour form with midnight as 00:00:00. The default time is midnight.

The AND operator provide the capability of concatenating several requirements on the files selected. If the same qualifier is specified more than once, only the last value entered for that qualifier is used.

The QUERY (or Q) switch instructs the ARCHIVE program to display the file name and then request user input before each file is copied or a destination directory created. The user must enter Y or y for the operation to occur. If the file is a directory file and the user does not answer Y or y, the directory is not created.

The UPDATE (or U) switch instructs the ARCHIVE program to delete duplicate copies of a data file automatically. The user is not queried regarding deletion.

Read access is required to copy source directory and data files. Add entry access is required to ADD files to existing destination directories. The Superusers, by default, can read all source data files and copy them to destination directories.

When the ARCHIVE command is given, the program signs on:

```
operating system ARCHIVE UTILITY, Vx.y
```

For each directory the screen displays the following information:

```
DIRECTORY = directory-name
```

After each copy the screen displays the following information:

```
COPY source-file TO destination-file
```

or, if QUERY is specified, the screen displays the following information:

```
COPY source-file TO destination-file ?
```

When the ARCHIVE program has executed, the screen displays the following information:

```
ARCHIVE COMPLETE
```

The source directory subtree structure, which is duplicated at the destination directory, includes the original owner, create time, and last time it was modified.

When a directory is encountered at the maximum supported subtree depth, a warning message is issued to the user and ARCHIVE continues, as shown in the following example:

```
UNABLE TO COPY SUBTREE - LEVEL TOO DEEP
PATHNAME = directory-pathname
```

You are responsible for archiving these additional subtree branches.

Examples

1. `ARCHIVE /source_dir TO /dest_dir Q`

   This example will copy all files with the proper access from /source dir to /dest_dir and will verify each operation with the user.

2.
   ```
   ARCHIVE /fatcat_dir TO /fatcity_dir
   FILES *.LST MODIFIED BEFORE 12/25/81
   ```

   This example copies all files in the directory subtree 'fatcat_dir' which match the pathname '*.LST' and which have been modified before December 25, 1981, to the destination directory 'fatcity_dir.'

CHOWNER

Syntax

`CHOWNER` *filename* `TO` *username*

where

username	is the name of the new owner of the file.
filename	is a fully qualified pathname or logical name.

Description

This function permits a change in ownership of a file. Only the superuser or the owner is authorized to use this command.

Examples

1. `CHOWNER/MEDIA TO PAUL`

   This example changes the ownership of the MEDIA file, giving it to user PAUL.

2. `CHOWNER/BOOK/ATEXT TO ART`

   This example changes ownership of the BOOT/ATEXT file, giving it to user ART.

CHPASS

Syntax

`CHPASS` *username*

where

username	is the assigned user's identifier.

Description

This command changes the password associated with the specified user. Any superuser may invoke CHPASS for any other user. However, the command will not execute for ordinary users unless the old password is correctly entered in response to the query produced. Superusers need not enter the old password.

Examples

1. `CHPASS FRED`

```
Old Password?
            pass1
New Password?
            pass2
Verify Password?
            pass2
```

In this example, user FRED (or any other ordinary user who knows FRED's old password) changes his password from pass1 to pass2. FRED then needs to verify that he knows the new password (pass2) by responding correctly to the third query (verify?). The new password (pass2) will not be displayed when entered at the keyboard.

COPY

Syntax

COPY *sourcefilename* TO *destinationfilename* $\left[\begin{Bmatrix}\text{UPDATE}\\\text{QUERY}\end{Bmatrix}\right]$

where

sourcefilename	is a pathname or a wildcard pathname. If the *source-filename* is a wildcard pathname, the *destination-filename* must be a directory file.
destinationfilename	is either an existing directory file or a data file.
UPDATE and QUERY	are options that suppress and enable the querying process, respectively.

Abbreviations

UPDATE can be abbreviated to U, QUERY to Q.

Description

The COPY command copies a file from one position within the file hierarchy to another position. If the *sourcefilename* is a wildcard pathname, the destination file must be a directory file.

The *destinationfilename* is either an existing directory file or an existing data file. If the *destinationfilename* is an existing directory file, a new file is created in that directory and the final component of its filename is the final component of the source filename. This operation cannot take place if you do not have ADD Entry Access to the directory file.

If the destination file is an existing data file (and the UPDATE option has not been specified, you are queried as shown in the following example:

```
FILE ALREADY EXISTS
PATHNAME = name of existing file
DELETE EXISTING FILE?
```

(In SUBMIT, only the first two lines will be printed; you will not be prompted to delete the file). Entering a Y (or y) deletes the existing file and initiates the Copy operation if you have Delete access. Any other entry aborts the operation. The UPDATE option disables the querying.

Querying can be explicitly chosen by entering the QUERY option. In this case, each Copy operation produces the following query:

```
COPY pathname TO pathname ?
```

Entering a Y (or y) causes the copy to take place. Any other entry terminates the command. The QUERY option cannot be used in submit mode; if QUERY is entered, it produces the following error message:

```
QUERY OPTION NOT USEABLE IN SUBMIT MODE.
```

Each successful copy produces the following message:

```
COPIED pathname TO pathname
```

You can also specify the Line Printer as the destination of the Copy operation, thereby allowing you to obtain printed output if the workstation has a local line printer. If the network workstation does not have a local line printer, the files may be spooled to a network printer. If the system does not have a local line printer, an error message will appear. To print output locally, specify :LP: as the *destinationfilename*. To provide for network spooling, enter :SP: *request name* as the destination. Copying to :CO: sends the file to the console output device.

NOTE

File protection attributes are not copied from file to file. With a successful copy, the owner of the destination file will have full access rights to that file.

CREATEDIR

Syntax

```
CREATEDIR pathname
```

where

pathname	is the pathname identifying the new directory.

Description

This command creates a new directory with the pathname entered only if you had ADD-Entry access to the parent directory and no existing file (either data or directory) already has that pathname. If a conflict in pathnames exists, the command will abort. Following proper execution of the command, you are given complete access rights to the file created. No access is conferred upon WORLD.

Examples

1. `CREATEDIR /DIRA/DIRB`

   A new directory, DIRB, is created in the parent directory, DIRA.

DELETE

Syntax

```
        ┌            ┌ DIR                       ┐ ┐
DELETE  │ pathname  <  QUERY                      >│
        │            └ :SP:    / request name    ┘ │
        └                                          ┘
```

where

pathname	is the pathname or the wildcard pathname or a null string.
DIR	is mandatory for deleting non-empty directory files and is optional for deleting empty directory files and all data files.
QUERY	is an option that produces interactive querying before each delete operation is executed.
:SP:	signifies a spool delete.
request name	is the name of the spool request to be deleted.

Description

The Delete command lets you delete both data files and directory files. In the command line, the keyword DIR is mandatory for deleting Non-Empty directory files and is optional for deleting Empty directory files. If the keyword DIR is used for data files (both empty and non-empty) an error message will be returned.

A given data file will be deleted only if you have DELETE access to that file.

In deleting a directory file, the following operations are performed: first, your access rights to the given directory file are checked. If you do not have Delete access, the operation terminates immediately; otherwise, every file in the directory will be deleted except:

1. Non-empty directory files (which are not deleted).
2. A file within the directory to which you do not have DELETE access.
3. Any file within the directory to which connections exist at the time the delete operation is requested. In this case, the directory is not deleted and no new connections to the file are permitted. The file will be physically deleted from the medium when the last connection to it is detached.

The specified directory will not be deleted if any of these conditions occur.

If the null string is entered as the logical directory name (i.e., DELETE <cr>), an attempt is made to delete the directory associated with the null logical name.

Spool requests are deleted by specifying the name of the spool request to be deleted. If no name is specified, the system will try to delete the request that is printing. With the exception of the Superuser (who may delete any spool request), a spool request may be deleted only by the owner.

The directory specified in the invocation will not be deleted if any file with that directory is being accessed by another user when the Delete command was attempted.

User Query Interaction

You may specify the QUERY option with any delete command.

In Interactive Mode: every delete (in the given delete invocation) produces the following query:

```
DELETE pathname ?
```

If you want to delete that file, respond by entering either Y or y followed by <cr>. If you do not want to delete that file, enter any other character.

In Submit Mode: the query option causes termination of the delete operation. The following message appears on the console:

```
QUERY OPTION NOT USEABLE IN SUBMIT MODE.
```

Each successful deletion of a file is followed by the following console message:

```
DELETED pathname
```

Successful directory deletion produces the following console message:

```
DELETED DIRECTORY pathname
```

Examples

1. `DELETE /A/*Q`

   This example deletes files in directory A with a query before each file deletion.

2. `DELETE /A/B/C`

   This example deletes File C in the Directory /A/B.

3. `DELETE /A/B DIR`

   This example deletes the directory file /A/B.
   Since /A/B is a non-empty directory file, the specifier DIR must be used.

Additional Notes

In a multi-programmed/shared file environment, certain time-dependent situations may cause a delete operation on a directory to fail. For example, if a file is created within a directory at the same time a delete operation is being performed on the directory, the file may not be encountered by the search operation and the delete will fail. If connections exist to one or more files within a directory at the time a delete is attempted, the delete may fail.

A directory will not be deleted if it is the parent of a non-empty directory file.

Wildcard pathnames may not be used with the DIR specification.

DIR

Syntax

$$\texttt{DIR}\ \left[\left\{\begin{matrix}\textit{pathname}\\ /\end{matrix}\right\}\ \texttt{[FOR}\ \textit{filename}\texttt{]}\ \texttt{[EXPANDED]}\ \texttt{[TO}\ \textit{pathname}\texttt{]}\right]$$

where

pathname	is either the pathname or a logical name.
FOR *filename*	specifies directory information of *filename* to be displayed.
EXPANDED	specifies that the completed information described below in the "Description" should be provided for the directory pathname entered.
TO	specifies a file where the completed information is to be written (in addition to the console).

Description

The Directory command allows you to display the names of files within a directory. When used without the expanded modifier, the Directory command produces a listing of file names only. Omitting the pathname produces a listing of the names of all files in the directory designated by the null logical name.

The EXPANDED modifier produces the following information:

- File name
- Owner name
- Length (in bytes)
- File Type (Data or Directory)
- Owner Access rights
- World Access rights

A null entry cannot be used with the EXPANDED or TO Modifier. To obtain expanded information on the file designated by the null logical name, you must enter " " as the directory name.

Information about the file system configuration can be obtained by performing a Directory operation on "/". For each volume of the file system which is accessible from the given node, the volume name and the volume location are returned. The location is specified as being Public, Local or Shadowed. A volume is Public if the WORLD can access it; use of that name as the first component of a fully qualified pathname results in a file on that volume being accessed (i.e., the public). A volume name is Local if only the owner can access it; use of the pathname as the first component of a fully qualified pathname results in a file on that volume being accessed. A volume is Shadowed if it resides in a shared file and a volume of the same name exists locally. The local volume shadows the public volume, i.e., use of this name as the first component of a pathname accesses the local volume, not the public volume.

Information about the state of the spool queue at a public node can also be obtained by using the Directory command. The spool is designated by the name SPOOL; DIR/*system device root directory*/SPOOL displays the contents of the spool.

For example, if WD0 is the system device directory, DIR/WD0/SPOOL displays the contents of the spool. The format of the display is the same as for file. The order of the files listed in SPOOL DIR is unrelated to the printing order in the print queue.

The Directory command sends output to the console as a default condition. However, the modifier TO can be used instead to designate any disk file or device (such as :SP: or *file*).

TO is subject to the restrictions of any other action that creates/updates a file, i.e., ADD-ENTRY, DELETE, WRITE ACCESS RIGHTS must be updated. Default rights on the new file will not necessarily be compatible with access rights in the original file.

Examples

1. `DIR /A EXPANDED`

   This example displays full information about all files in directory A.

2. `DIR /A`

   This example displays only file names for directory A.

3. `DIR /A TO /A/B`

   This example outputs file names of directory A to file /A/B.

4. `DIR /SYS.DEVICE/SPOOL`

   This example displays files in SPOOL Directory if the system device root directory is SYS.DEVICE.

5. `DIR /DIR FOR PROGA.SRC EXPANDED`

   This example displays expanded information about the file PROGA.SRC, which resides in the directory WORK.DIR.

6. `DIR /WORK.DIR FOR *.SRC`

   This example displays all filenames in the directory WORK.DIR which have the extension .SRC.

Additional Notes

Wildcard pathnames cannot be used with the Directory command.

DIR / EXPANDED is not a valid command.

DISMOUNT

Syntax

`DISMOUNT` *device name*

where

device name	is: FL0, FL1 for 5-1/4″ Flexible Disks WM0 for the integrated Winchester 5-1/4″ Disk WD0, WD1 for a Winchester Priam 35-Megabyte Disk HD0 thru HD3 for HD5440 Hard Disks

Description

This command makes a mass storage device inaccessible to the file system and detaches all connections previously made to the file. The Dismount program guarantees that the specified volume will no longer be accessed by the file system. Never dismount the system volume. An error message is returned if the specified device has already been dismounted.

Examples

1. `DISMOUNT HD0`

   This example removes the hard disk from the file system.

2. `DISMOUNT WD0`

   This example removes the Winchester disk from the file system.

LNAME

Syntax

To CREATE a Logical Name:

```
LNAME DEFINE logical name FOR pathname [UPDATE]
```

where

logical name	is a user defined string of up to fourteen characters which you use to reference a directory. A logical name has the same syntax as a fully qualified pathname.
pathname	is the pathname for a directory.
UPDATE	is an option that automatically executes the command even if the logical name has previously been assigned; UPDATE refers the logical name to the new pathname.

To DELETE a Logical Name:

```
LNAME REMOVE logical name
```

where

logical name	is the user defined string used as a logical name.

To DISPLAY a Logical Name:

```
LNAME PATH
```

where

PATH	displays each logical name and its associated path component.

Description

Logical names are user-defined character strings that are equivalent to pathnames. Logical names are easier to remember than pathnames and often have greater meaning to the user. The Define Logical Name command assigns a logical name string to a pathname. If the logical name chosen conflicts with an already existing logical name, you are queried:

```
LOGICAL NAME ALREADY EXISTS, REDEFINE?
```

Entering a Y (or y) causes the command to be executed and causes the existing logical name to be redefined. Any response other than Y (or y) aborts the command. The UPDATE option displays this querying.

Examples

1. `LNAME DEFINE X FOR /A/B`

   The directory specified by the pathname /A/B is assigned the logical name X and can be accessed as such.

2. `LNAME REMOVE DATA`

   The logical name DATA is deleted. The directory previously accessed by this logical name must now be accessed by its pathname.

3. `LNAME DEFINE " " FOR /FILE/A/C`

   The null logical name is assigned to the directory /FILE/A/C. That file can now be accessed simply by entering a space. Assume that data files /DATA1 and /DEBT are in the directory file /FILE/A/C. After defining the null logical name for /FILE/A/C, you can access the data files as simply (space) DATA1 and (space) DEBT. To obtain an expanded directory of /FILE/A/C, you must enter DIR " " EXPANDED, not DIR EXPANDED.

4. `LNAME PATH`

   Each logical name and its associated pathname is listed.

Additional Notes

Logical names may only be defined for directory files.

Logical names are defined for the duration of a log-on session only; they do not transfer to background and remote jobs. When you log off, the logical names created during that session are deleted. In the next session, you must redefine all the logical names.

MOUNT

Syntax

```
MOUNT device-name
```

where

device-name	is: FL0, FL1 for 5-1/4″ Flexible Disks
	WM0 for the integrated 5-1/4″ Winchester Disk
	WD0, WD1 for a Winchester Priam 35-Megabyte Hard Disk
	HD0 thru HD3 for a Fixed Platter Hard Disk

Description

The Mount Command allows you to explicitly request a device to be mounted; each device represents a different volume in the iNDX file system. The Mount command enters the device's volume root directory into the system root directory. If the name of the volume root directory being entered conflicts with an existing volume name of the local file system, the new disk will not be mounted. If the new volume name conflicts with a public file name, the mount will take place because the new volume name "shadows" the existing public volume name. When you enter a pathname that contains the volume name as a component, the system will access the local filename. The public volume name is effectively hidden from you and is not accessible.

The mount operation may be performed only on mass storage devices. Unless a device has been operationally dismounted (DISMOUNT), you can usually access it without using the MOUNT command. The floppy volume is mounted when it is inserted into the drive. Hard disk volumes are mounted when the device is ready after power-up. If the Mount operation is requested on an already mounted device, the following message is displayed:

```
DEVICE ALREADY MOUNTED
```

This is not an error condition; the device will remain mounted.

The file structure on the volume being mounted is integrated into the existing directory structure by logically adding the volume root directory as an entry in the system root directory.

Examples

1. `MOUNT WD0`

   This example mounts the 35-Megabyte Winchester disk.

2. `MOUNT HD1`

   This example mounts the removable hard disk.

RENAME

Syntax

```
RENAME old-pathname TO filename [UPDATE]
```

where

old-pathname	is the pathname that presently identifies the file.
filename	is the last path component that you now want to identify the file.
UPDATE	disables the user interactive querying.

Description

The RENAME command renames a file by changing the last element of its path to the component specified by the *filename*. The *filename* must be a single path component. To use the RENAME command you must have DELETE access to the given file.

If the new name to be assigned to the file conflicts with an existing filename, you will be queried as follows:

```
PATHNAME = pathname
DELETE EXISTING FILE?
```

where

pathname	is the conflicting pathname.

If you respond to the query by entering a Y (or y), the existing file will be deleted and the Rename will take place (provided you have proper access rights). Any response other than Y (or y) will abort the command and the existing file will not be deleted. The UPDATE option disables the querying process. If the conflicting file is a Directory file, the Rename cannot take place and the following message will appear on the console:

```
FILE ALREADY EXISTS
NAME CONFLICTS WITH EXISTING DIRECTORY FILE
PATHNAME = pathname
```

Examples

1. `RENAME /X/Y TO Z UPDATE`

   File /X/Y is renamed to Z. You are not queried before the change takes place.

Additional Notes

1. Wildcard pathnames cannot be used as the new pathname.
2. Directory files cannot be deleted as the result of a Rename.
3. In Submit Mode, failure to specify the Update option results in an error message. The Update option should always be used in Submit Mode.
4. When multiple users can access the same file, using the Rename command can prevent other users from accessing that file.

SDCOPY

Syntax

```
                                              ⎧FORMAT⎫     ⎡       ⎤
SDCOPY source-device-name [TO dest-device-name]⎨REPEAT⎬ ... ⎢COMPARE⎥
                                              ⎩VERIFY⎭     ⎣REPEAT ⎦
```

where

source-device-name	is the source device, i.e., either FL0 or FL1.
dest-device-name	is the destination device, i.e., either FL0 or FL1.
FORMAT	is an option that first formats the destination device.
VERIFY	is an option that verifies the destination device after it has received the copy of the source.
COMPARE	is an option that compares the destination to the source to see if they are the same.
REPEAT	is an option that repeats the preceding operation (i.e., option) using the same source.

Abbreviations

FORMAT can be abbreviated to F, VERIFY to V, COMPARE to C, and REPEAT to R.

Description

SDCOPY allows you to duplicate 5-1/4" flexible diskettes using a single drive system. SDCOPY provides track-for track copies of one disk to another with options that permit formatting the destination device and/or verification after the copy is made. Another option allows two diskettes to be compared for equality. SDCOPY cannot operate in Submit mode.

User Interaction

After you have entered the SDCOPY command, the following message will appear:

```
operating system ID  SDCOPY,  Vx.y
PLEASE LOAD  type DISK INTO device name
Type <CR> to continue, E to exit
```

where *type* is either SOURCE or DESTINATION.

When SDCOPY has completed an operation, the following confirmation message will appear:

```
SDCOPY COMPLETE
```

If the VERIFY or the COMPARE options were selected, one of the following messages will appear (depending on the results of the operation):

```
VERIFICATION OK
```

or

```
VERIFICATION FAILURE
```

SDCOPY assumes it has only one drive to work with unless you specify the optional TO clause. The single drive configuration is used when no TO clause is entered or is used if the destination specified in the TO clause is the same as the source. If the device-name is that of a system device, SDCOPY aborts and the following message appears:

```
CANNOT COPY DISKS USING A SINGLE SYSTEM DEVICE
```

If a Winchester disk is available, SDCOPY will use it as temporary storage. The entire source disk will be copied to the Winchester and only a single operation is required to accomplish the copy. The number of disk swaps is calculated as the total size of the source disk divided by the size of the work buffer. This number will be displayed and the following prompt requesting permission to continue will appear:

```
IT WILL BE NECESSARY TO SWAP DISKS number TIMES.
DO YOU WISH TO CONTINUE? ( [Y] or [N] )
```

Any response other than N (or N) is treated as Y, i.e., permission granted.

If the TO option is specified with different devices for destination and source, no disks swaps are necessary. For example, with FL0 as the source and FL1 as the destination, the following message will appear:

```
PLEASE LOAD SOURCE DISK INTO FL0
PLEASE LOAD DESTINATION DISK INTO FL1
Type <CR> to continue, E to exit
```

Any entry other than E (or e) is taken as permission to continue.

If you specify the REPEAT option, the same source may be used for multiple operations without requiring you to reload the source before each operation. This option is useful if a Winchester disk can be used as a workfile or if two operable flexible disks are available.

Possible Error Messages

All I/O and syntax errors that take place during SDCOPY operation are fatal. In case of a syntax error, the correct syntax will be displayed and the command will terminate.

Verification errors are recoverable. Operation will continue after one of the following errors is displayed:

```
VERIFY FAILED AT TRACK track number SECTOR sector number
```

or

```
MISCOMPARED AT TRACK track number SECTOR sector number
```

Additional Notes

1. Duplicating disks using a single system device is not allowed because the system device cannot be dismounted if the system is to remain operational.
2. SDCOPY can only execute in the Foreground. Failure to run SDCOPY in the Foreground will result in command termination and the following message:

   ```
   CANNOT RUN IN BACKGROUND PARTITION.
   ```

3. SDCOPY cannot be run in Submit mode because it queries the user. If you attempt to run SDCOPY in Submit mode, the command will terminate and the following message will appear:

   ```
   CANNOT RUN UNDER SUBMIT MODE.
   ```

   In BACKGROUND mode the message is:

   ```
   CANNOT RUN CONCURRENTLY WITH A BACKGROUND JOB.
   ```

4. SDCOPY allows you to change the name of the volume root Directory on the Destination Flippy Disk. When the copy is completed (Prior to SDCOPY complete message), SDCOPY will ask the following question:

   ```
   DO YOU WANT TO CHANGE THE VOLUME ROOT DIRECTORY NAME?
   ```

   A response of y or yes (uppercase or lowercase) will result in the following message:

   ```
   CURRENT VOLUME ROOT DIRECTORY NAME IS XVOL
   PLEASE ENTER NEW VOLUME ROOT DIRECTORY NAME
   ```

   You can enter a new volume root name of up to 10 characters.

SPACE

Syntax

SPACE */ volume-name*

where

volume-name	is the volume root directory for the given physical device.

Description

The Space command returns information about the amount of available space on a given disk at that time. Information is returned in the following format:

```
VOLUME GRANULARITY = number (granularity = number of bytes/
                            sector)
FREE BLOCKS = number
TOTAL BLOCKS = number
FILES AVAILABLE = number
TOTAL FILES = number
```

MM/DD/YY hh:mm:ss

where

MM/DD/YY hh:mm:ss	is the month, date, year, hours, minutes, and seconds.
number	is a decimal integer.

Space will not accept logical names as input—volume names must be used.

Examples

1. SPACE /A

   The information is provided for the volume whose root directory is A.

USERDEF

Syntax

```
USERDEF { DEFINE username[ID userID] [DIR filename] }
        { REMOVE username                           }
```

where

username	is the name by which the system identifies the user.
userID	is the ID-number by which the system identifies the user.
filename	is the name of the user's home directory.

Description

This function allows the superuser to add or delete a user who has access to files on mass storage devices. An attempt to create a new user will fail if the specified user name or user ID is already used. When creating a new user, the superuser has the option to simultaneously create a directory for that user. Likewise, the directory creation will not be successful if a file with the same name already exists. When a directory is successfully created, the newly created user owns it and has full access rights to it.

The superuser has a predefined name within the system—SUPERUSER. This name is used by the primary superuser and cannot be deleted. The primary superuser may create secondary superusers with the USERDEF function by specifying a user ID in the allowable range (3-15). (Other valid user IDs are in the range 1024-32767.) Secondary superusers have full superuser capabilities; however, they may not execute the USERDEF and USERS commands, and they may be deleted from the system by the primary superuser.

When a user is deleted from the system, only the user name is deleted. If that user is logged on the system, that session continues. All files that belong to the deleted user remain in the system. An expanded directory listing returns a user name of NOT FOUND for those files. You must use the CHOWNER command to transfer ownership of these files.

A list of all users and user IDs assigned via the USERDEF command can be obtained through the USERS command. For each user created, you should create a directory on an available volume and tell that user the fully qualified pathname for that directory. The user should place all of his subsequent directories and files on the directory you assigned.

Examples

1. `USERDEF DEFINE PAUL ID 1025`

   This example adds user name PAUL with ID 1025.

2. `USERDEF DEFINE SHEILA ID 32732 DIR /A/B`

   This example adds user name SHEILA with ID 32732 and a directory named /A/B.

3. `USERDEF REMOVE BARRY`

   This example removes user name BARRY.

4. `USERDEF DEFINE LINDA DIR /FINANCE/ACCT`

   This example gives the existing user LINDA a directory named /FINANCE/ACCT.

Additional Notes

1. The initial password for each user is " "(null). You should immediately use CHPASS to assign a password after assigning a user name.
2. The USERDEF command manipulates two system files: UDF and HOME. These files are located in the volume root directory of the system volume. Do not change them.

NOTE

Back up of the UDF and HOME files should always be done simultaneously to ensure system consistency.

3. USERDEF can be used to change the home directory of an existing user by entering the following command:

   USERDEF DEFINE *existing username* DIR *new filename*

USERS

Syntax

```
USERS /volume-name
```

where

volume-name	is the volume root directory name.

Description

This function lists in tabular form the user names and associated user IDs for all users who have log-on rights to mass storage devices. The *volume-name* specified should be the volume root directory name of the system volume.

NOTE

This command can only be used by the superuser; the secondary superuser has no access rights to it.

Examples

1. `USERS /ALPHA`

   This example requests a list of the users on volume ALPHA. The following list is displayed:

```
USERNAME   USERID

PAUL       1025
SHEILA     32723
```

VIEW

Syntax

```
VIEW pathname
```

where

pathname	is a valid pathname.

Description

The View command is an interactive command that allows you to examine the contents of the specified file. The View command must be entered from the keyboard and cannot be placed in a command file. When the View command is entered from the keyboard, the specified file is attached and its first page is displayed in AEDIT full screen format. The text of the page appears on Lines 1-23 of the display. Line 24 (the AEDIT message line) displays "----VIEW pathname", where pathname is the name of the file being read. Line 25 (the AEDIT prompt line) displays the subset of AEDIT commands available for use with the View command. The available commands are: Again, Find, -Find, Jump, Quit, Roll, Set, View.

Job Management Commands

Following are the job management commands.

BACKGROUND

Syntax

```
                                              ┌┌   LOG ┐┐
BACKGROUND pathname [(a-parameters)] . . .    ││       ││
                                              └└ NOLOG ┘┘
```

where

pathname	is a valid pathname.
a-parameters	is a list of up to 10 parameters.
LOG and NOLOG	specify whether a log is to be kept on a mass storage of all console activity.

Description

The Background command, used to execute a command file on a workstation, permits the simultaneous execution of a job requiring user interaction. The difference between executing a given command file in the background and executing a given command in the foreground is that background execution does not allow console interaction. You may use the LOG option to provide a log-on to a mass storage device. The Background, Export and Submit commands all have similar structures. The similar structures permit you to execute the same command file in any of these modes.

If the name of your command file does not have an extension, the system appends the extension .CSD to the command file. If your filename ends with a period, the system will use the filename you have given (with the period truncated). If your filename includes an extension, the system uses that filename without modifying it.

The optional parameters specified in the command line are the actual parameters to be substituted for the formal parameters embedded within the command file. A maximum of ten actual parameters may be specified in the command line. Placing formal parameters in the command file allows you to call the same command file for varying sets of actual parameters. When no parameters are specified in the command line, the specified command file is executed directly. When parameters are specified, a second command file will be generated in which the actual parameters are sequentially substituted for the formal parameters. Thus (reading from left-to-right), if you enter the actual parameters ASM, PLM, and BLT, ASM replaces the formal parameter %0, PLM replaces the formal parameter %1, and BLT replaces the formal parameter %2. The command file resulting from the parameter substitution is placed in the same directory as the command file from which it was generated. The final path component of the generated file is derived from the final path component of the generating file as follows:

- The final component of the generating file after the fifth character or before the first period (whichever comes first) is truncated.
- A unique character string obtained from the system job manager and the extension .TMP are appended.

When the LOG option is specified, a log of what would normally appear on the console is kept on a mass storage device if the job is executing in the foreground. When LOG is specified, the log file gets the extension .LOG. NOLOG specifies that no log be kept. If neither option is explicitly entered, LOG is the default condition.

When a background job is executed, the system implicitly re-enacts your LOGON. The environment created is the same one that existed when you initially logged on to the system. This environment is not necessarily the same as the foreground environment operating at the time the Background command was invoked. The logical names defined in the foreground are not accessible in the background and vice versa. The amount of memory available for the background is specified by the Region command.

Examples

Refer to the examples given under the Submit command for a comprehensive set of examples that illustrate the manner in which command files are generated.

CAUTION

1. Do not debug in the foreground while background programs are running.
2. Never run experimental software in the foreground while programs are running in the background.
3. Never run experimental software in the background or export it to another workstation.

BATCH

Syntax

`BATCH` *pathname*

where

pathname	is a valid pathname.

Description

The Batch command allows you to interactively create a command file using the Syntax Guide. If the name of your command file does not have an extension, the system appends the extension .CSD to the command file. If your filename ends with a period, the system will use the filename you have given (with the period truncated). If your filename includes an extension, the system uses that filename without modifying it.

A search is made for the specified command file. If the specified command file exists, it is read in and the Syntax Guide is invoked so you can modify the file. If the command file does not exist, it is created and then the Syntax Guide is invoked. When you have finished using the Syntax Guide to edit the command file, press the Escape key (ESC). The following prompt will appear:

(Line 24) — `SELECT EXECUTION OPTION`
(Line 25) — `Abort Write Submit Background Export`

Select the menu entry you desire by either pressing the Function key associated with that entry (e.g., FO selects Abort, F4 selects Export), or by entering the uppercase letters for that entry at the keyboard (e.g., A selects Abort). The entries have the following effects:

Abort—Do not save the edited command file. Return to the Command Line Interpreter (CLI).

Write—Write the edited command file. Return to the CLI.

Submit, Background, and Export—If Submit, Background, or Export is selected, you are prompted:

`WRITE` *command file*? `(y or [n])`

A response of y (yes) causes the edited command file to be saved before being executed. The default response of no will not save the edited command file.

You will also be prompted to check for parameters in the command file. If no parameters are present, select execution mode.

Parameter substitution: for each formal parameter in the command file, respond to each prompt as shown in the following examples:

Prompt: `ENTER VALUE %`*n*

where

n	sequentially takes the values of 0-9 for each formal parameter in the command file.

Response: For each parameter, enter a parameter name of up to 36 characters terminated by Escape or <cr>. The completed parameter name now appears on the message line (Line 24) of the display screen.

After you have entered all of the parameters, the following prompt, which requests verification of the parameter values, appears.

Prompt: `ARE THE PARAMETERS CORRECT?`

Response: If all of the parameters are correct, enter Yes (Default); if not, enter NO. The system replies to the No response as follows:

Prompt: `WHICH PARAMETER IS INCORRECT?`

Response: Enter the number of the incorrect parameter (0-9). If the number is valid, a prompt will appear. The prompt requests you to enter the correct value of that parameter. If the number was valid (i.e., it did not have a formal parameter in the command file corresponding to it), the number prompt returns. After the correction has been made, the prompt "ARE THE PARAMETERS CORRECT?" reappears. Respond accordingly.

The parameter substitution is now performed and the resulting workfile created. The pathname for this file is generated according to the rule described under the Submit command. If you initially selected the menu entry Export (i.e., the command file is to be executed at a remote workstation), a prompt will appear. This prompt asks for the name of the queue where the job is to be sent.

Another prompt will appear asking you if you want a log kept of console activity. Enter y for yes, N for No. The default condition for Submit is No; for Background and Export the default is Yes.

The command file is now executed according to the execution mode you selected earlier. If you selected Foreground execution, execution would take place precisely as it would for a Submit command. If you selected Background, execution would take place precisely as it would for a Background command. If you selected Export, execution would take place precisely as it would for an Export command. See the Background, Export and Submit commands for further information.

CANCEL

Syntax

```
CANCEL { BACKGROUND
       { REMOTE      queue { (jobname)  (#jobnumber) } }
```

Description

The Cancel command is used to cancel either a background job or a remote job (see the Cancel command in Chapter 5). If BACKGROUND is specified, thc currently executing background job is aborted.

REGION

Syntax

```
REGION
```

Description

The Region command is used to adjust the size of the regions and the binding between regions and jobs. In Foreground/Background systems, the free space is divided into two regions: Region 1 and Region 2. The default condition is for Region 1 to be allocated 96K of free space and for Region 2 to be allocated the remainder of the free space. Region 1 begins immediately above the operating system and Regions 2 begins immediately above Region 1.

A foreground job may use either Region 1 or Region 2. Background jobs then use the remaining region. If you are not executing 8-bit programs, it does not matter whether the Foreground job runs in Region 1 or 2. If 8-bit programs are frequently executed however, the binding between jobs and regions determines if 8-bit programs can be executed in the top 64K of user memory (the region where the ISIS parameter is true—the IEU memory space). Thus, 8-bit programs are usually executed in Region 1.

When the Region command is invoked, it displays the current size of each region, the binding between jobs and regions, the status (whether 8-bit programs can be executed in that region) and the starting address of each region. A prompt will appear and will ask you if you want to make changes or exit the command. If you want to make changes, you will be further prompted for the appropriate entry to change. After the change is entered, an updated display will appear, as will a request for verification of the updated display. Since both regions share a fixed amount of space, any change to one region produces an offsetting change to the other region. Thus, if you reduce Region 1, you automatically increase the size of Region 2.

If only one job is running in the foreground, the system temporarily combines the two regions into one and allocates the combined region entirely to the foreground. Thus, it is not necessary to change the region size frequently to achieve maximum utilization of memory. This also ensures that if no background job is currently running, an 8-bit program can always be executed in the foreground.

The Region command can only be invoked from a foreground job.

SUBMIT

Syntax

```
SUBMIT pathname [(a-parameters)] ... [{ LOG  }]
                                      [{NOLOG}]
```

where

pathname	is a valid pathname.
a-parameters	is a list of up to 10 actual parameters to be substituted during execution for the formal parameters embedded within a file.
LOG or NOLOG	specifies whether a log is to be kept on the mass storage device.

Description

The Submit command is used to execute a command file that does not require user interaction in the foreground of the local workstation. If the name of your command file does not have an extension, the system will append the extension .CSD to the command file. If your filename ends with a period, the system will use the filename you have given (with the period truncated). If your filename includes an extension, the system will use that filename without modifying it.

The option parameters specified in the command line are the actual parameters to be substituted for the formal parameters embedded within the command file. A maximum of ten actual parameters can be specified in the command line. Placing formal parameters in the command file lets you call the same command file for varying sets of actual parameters. If no parameters are specified in the command line, the specified command file will be executed immediately. When parameters are specified, a second command file will be created and the actual parameters will be sequentially substituted for the formal parameters in that file. Thus, reading from left-to-right, if you enter the actual parameters ASM, PLM, and BLT, ASM will replace the formal parameter %0, PLM will replace %1, and BLT will replace %2. The resulting command file will be placed in the same directory as the generating command file. The resulting command file's final path component is derived from the final component of the generating file as follows:

- The final component of the generating file after the fifth character or before the first period (whichever comes first) is truncated.
- A unique character string obtained from the system job manager and the extension .TMP are appended.

When the LOG option is specified, a log with the extension .LOG is written to a mass storage device. The NOLOG option specifies that no such log is to be kept. If neither option is explicitly specified, NOLOG is the default condition.

Examples

1. `SUBMIT 1/CMDFILE LOG`

   Since the filename in the command line does not contain an extension or terminate in a period, the extension .CSD is automatically appended. The file executed takes the name 1/CMDFILE.CSD. Since the LOG option has been specified, a log file is created under the name 1/CMDFILE.LOG. Assume that 1 is a logical name that has been assigned to a pathname.

2. `SUBMIT 1/CMDFILE. LOG`

   Since the filename in the command line terminates in a period, the period is truncated and the name of the executed command is 1/CMDFILE. A log file is created under the name 1/CMDFILE.LOG.

3. `SUBMIT 1/CMDFILE.XXX LOG`

   Since the filename in the command line has an extension, a log file is created with the name 1/CMDFILE.LOG.

4. `SUBMIT 1/CMDFILE (VRD, 1) LOG`

   Since the filename does not contain an extension or end with a period, the extension .CSD is automatically appended. Since the parameters VRD and 1 have been specified, a new file in which these actual parameters replace every occurrence of the formal parameters is generated. This file is named 1/CMDFI1E.TMP as the CMD FILE is truncated to five characters and the number I01E is appended by the system as the extension .TMP. Note that the number I01E and similar strings used in the following examples is simply a sample number. The precise string used is determined by the state of the operating system at that time. The strings have no effect upon order of execution or sequence of assignment. Within a given log on session, up to 4,096 (36^3 for remote jobs) unique numbers can be assigned. When that limit is reached, the numbers are assigned all over again. Also note that in the example the system assigned number is used in forming the name of the log file 1/CMDI01E.LOG.

5. `SUBMIT 1/CMDFILE. (VRD, 1, DBG)`

   Since the input filename terminates in a period, the peiod is truncated to produce the filename CMDFILE. Because parameters /VRD, 1 and DBG have been specified, a file must be generated in which these actual parameters can be substituted on a one-for-one basis with the formal parameters %0, %1, and %2. Note that a file name, a symbol and a number are all acceptable actual parameters. The parameter substitutions are made into a file named 1/CMDI020.TMP because the file name has been truncated to five characters and the system number and extension .TMP appended. The log file is created under the name 1/CMDI020.LOG.

6. `SUBMIT 1/CMDFILE.XXX (/VRD,1) LOG`

   Since an extension has been specified in this example, no extension is appended—nor does truncation occur. The command file form in which substitutions are made is 1/CMDFILE.XXX. Because parameters have been specified, the substitutions of the actual parameters for the formal parameters are made to the file 1/CMDI022.TMP. CMDFILE.XXX has been truncated to five characters and the system assigned number and the extension .TMP have been appended. The log file is created under the name 1/CMDI022.LOG.

Additional Notes

If the same command file is simultaneously executed in the foreground and in the background, and no parameters are specified in the Submit command, only one log file will be produced. This occurs because even if the LOG option is selected for both jobs, the naming rules produce two identically named log files. For example, if you enter the following two commands:

```
SUBMIT 1/A LOG
BACKGROUND 1/A LOG
```

both will request log files having the file name 1/A.LOG. If the order of Background and Submit were reversed, the conflict would still occur. To prevent this conflict, always enter a null parameter for the Submit command if the same file is being simultaneously executed in Background mode. Thus, by using SUBMIT 1/A ( ) LOG you would produce a file for the Submit command called 1/A. I001E.TMP even though parameters to be substituted are nonexistent. The log file then takes the name 1/AI01E.LOG.

Media Operation Commands

Following are the media operation commands.

FORMAT

Syntax

```
FORMAT  physical-device volume-name [{FNODES(number) NOINIT
        RESERVE (reserve-option,...)}] ...
```

where

physical-device	is: FL0, FL1 is a flexible disk WM0 is a 5¹/₄″ Winchester disk WD0, WD1 is an 8″ Winchester disk HD0 is a fixed platter hard disk HD1 is a removable platter hard disk
volume-name	is the volume root directory name of the physical device.
reserve-option	is: OS (number)—operating system OV (number)—overlay

Description

The FORMAT command formats a disk volume and initializes it with the volume operations files. These files do not include system programs.

The FORMAT command verifies each block on the disk. Any block that cannot be read is marked in the free space bit map and the bad block bit map to prevent the block from being allocated to a file. If the bad block is in the fixed area reserved for the volume label, the FORMAT command terminates after displaying the appropriate message.

The FNODES parameter specifies the number of filenames to be reserved. If the number is not specified, a default value is used:

Device	Minimum	Default	Maximum
FLn	16	200	400
HDn	16	1000	2000
WDn	16	3000	6000
WM0	16	2000	4000

The NOINIT parameter specifies that only the file structure is to be initialized. This option, used to reformat disks that have been previously formatted, speeds up the formatting process because the formatting of disk sector IDs is not performed. FORMAT will override the NOINIT keyword when the specified disk is one that has not been previously formatted.

The RESERVE keyword specifies that one or more of the specified files are to be created and that contiguous blocks are to be allocated to them. FORMAT does not write data into these file blocks; SYSGEN writes into these files. The number of reserved blocks, not to exceed 1000 blocks, may be explicitly specified. The default numbers are 256 for the operating system and 192 for the overlay.

All workstation users should log off or refrain from accessing the target disk while you are formatting any disk other than a flexible one. If a file on the target disk is open at a workstation when the FORMAT command is entered, subsequent operations on the file will return an error indicating that the disk has been dismounted. The file on the target disk that was open will be destroyed.

Examples

1.
```
FORMAT FLO ONE.VOL.
```

   This command will format the flexible disk volume and name it ONE.VOL. When the return is entered, the following message will appear on the screen:

```
iNDX-system-ID FORMAT, Vx.y
```

FPORT

Syntax

```
         ⎧UP    iNDX-source-pathname TO destination-pathname⎫ ⎧UPDATE⎫
⎧S4FPRT⎫ ⎨EXIT                                               ⎬ ⎨EXIT  ⎬
⎩S2FPRT⎭ ⎪DOWN  [disk-dir] ISIS-source-pathname              ⎪ ⎪      ⎪
         ⎩      TO iNDX-destination-pathname                 ⎭ ⎩QUERY ⎭
```

where

S4FPRT and S2FPRT	inform the operating system that you are initiating the command from a Series IV or Series II, respectively.
iNDX-source-pathname	is a valid iNDX pathname or wildcard pathname.
destination-pathname	is a disk-directory or an ISIS-filename (optionally preceded by a disk-directory). ISIS-filename and disk-directory are as defined in the *ISIS-II User's Guide*, 9800306.
disk-dir	is a disk-directory as defined in the *ISIS-II User's Guide.*
ISIS-*source-pathname*	is an ISIS-filename or an ISIS-wildcard-filename as defined in the *ISIS-II User's Guide.*
iNDX-destination-pathname	is a valid iNDX directory file or a valid Series-IV pathname.
UPDATE, EXIT, and QUERY	are options that determine if you are to be queried prior to each copy operation.

Description

The FPORT utility program allows you to copy ISIS files to a specified location within the iNDX file structure and vice-versa. The copy is accomplished by transmitting data through a serial line connecting the Series-IV system and the ISIS system. While the FPORT program is executing, the ISIS system acts as a slave of the Series-IV system and waits for commands to reach it from the Series IV. Either the Exit function of FPORT or the EXIT option must be used to return the ISIS system to independent operation.

The FPORT command has three functions: UP, EXIT, and DOWN. Each function has its own set of qualifiers.

Physical Interaction

To execute the FPORT command, you must first boot up the ISIS operating system on the Series II system and then enter the ISIS-II File Port program. The program will then return the following message and enter a wait state:

```
ISIS-II FILE PORT, Vx.y
ENTER FPORT COMMAND at the SERIES-IV CONSOLE
```

From this point on, enter all commands only from the Series-IV keyboard.

Copying iNDX Files to ISIS Files

The UP function allows you to copy an iNDX file to an ISIS file. If you specify an iNDX wildcard-pathname as the iNDX source-pathname, the destination-pathname must be a disk-directory. If the iNDX source-pathname is not a wildcard-pathname, the destination-pathname may be either an ISIS-filename or a disk-directory or both.

If the destination-pathname is a disk-directory, the constructed ISIS file takes the name of the source file. If the file name is invalid under ISIS, an error message is returned. If the destination-pathname conflicts with the name of an existing file, you will be queried as to whether you want to delete the existing file and copy the source file to that file name. You can delete an existing file by writing over it only if the file's write and format attributes are not set. By using the UPDATE option, you can disable the querying process.

Copying ISIS Files to iNDX Files

The DOWN function allows you to copy an ISIS file to a specified location within the iNDX hierarchy. If the ISIS source pathname is an ISIS wildcard filename, the iNDX destination pathname must be an iNDX directory name. If the ISIS source pathname is not a wildcard filename, the iNDX destination pathname may be either a data file or a directory file.

If the iNDX destination file exists and is a directory file, a file of the same name as the ISIS source file is created within the iNDX directory file if you have ADD-entry access to that directory file. If the pathname constructed for the file exceeds the iNDX limit of 127 characters, an error message is returned.

If the iNDX destination file exists and is a data file, you will be queried as to whether you want to delete the existing file and assign that name to the new file. Such deletion can occur only if you have DELETE access for the existing file. You can suppress the querying by specifying the UPDATE option in the control line.

If the destination filename does not already exist, a new file of the specified name is constructed within the parent directory if you have ADD-entry access to the parent directory.

Returning the ISIS System to Independent Operation

During execution of the FPORT command, the ISIS system acts as a slave of the iNDX system. To return the ISIS system to independent operation, you must use either the Exit option or the EXIT function. If you have a number of files to copy, enter the Exit option after you have copied the final file. If you have only one file to copy, enter the EXIT function in the command line and the ISIS system will return to independent operation as soon as the file has been copied.

Transmission of the file begins when you terminate the command line on the Series-IV system. When you terminate the command line, FPORT presents the following sign-on message:

operating system name `FILE PORT, V`*x.y*

FPORT then performs a syntax check and checks the validity of the pathnames you have entered. If any errors are detected, an error message is generated and the command is aborted. In this case, you must re-enter the entire command.

User Interaction

If no errors are detected, the querying process begins. If the destination filename already exists, you will be queried to see if you want to delete the existing file by copying the new file to this filename. The following message will appear if you are in the interactive mode—

```
FILE ALREADY EXISTS
PATHNAME is pathname
DELETE EXISTING FILE?
```

If you respond by entering a Y (or y) followed by <cr>, the command proceeds and the existing file is deleted, (if you have delete access rights to the file). Any response other than Y (or y) causes the command to go on to the next file to be copied, to repeat the process if a wildcard pathname were selected, and otherwise terminates the command successfully. Since your console responses are line edited, you must enter a line terminator (<cr>) after each response.

If you specify the UPDATE option in the command line, the querying process is suppressed. The QUERY option may be specified by invoking FPORT. In interactive mode, the QUERY option causes the following query to appear before each file is copied:

```
COPY pathname TO pathname?
```

If you respond by entering a Y (or y) followed by <cr>, the FPORT operation will execute. Any other response causes the operation to go on to the next file to be copied (in the case of a wildcard pathname) and otherwise terminates unsuccessfully. Enter a line terminator after each console response or the response will not be accepted.

In Submit Mode, the QUERY option is not allowed; if the QUERY option is attempted, the following error message will appear:

```
QUERY OPTION NOT USEABLE IN SUBMIT MODE
```

Each successful FPORT copy operation results in the following message:

```
COPIED pathname TO pathname
```

If at any time the expected data is not received by either system, the following error message will appear:

```
DEVICE TIMEOUT ERROR
```

All operations will terminate and both FPORT programs (ISIS and iNDX) will be aborted. (Both programs loop while waiting to receive responses from the other system; if no response is received, the FPORT program will terminate (timeout) and control will return to ISIS or iNDX.)

Examples

(Observe the prompts at the Series-II system and enter the responses shown below.)

1.
```
-FPORT

SERIES-II FILE PRT, Vx.y

ENTER FPORT COMMAND at the SERIES-IV CONSOLE
```

(Enter the commands at the Series-IV system and observe the console messages shown below.)

2.
```
FPORT DOWN :F1:A TO /A/B
```

```
COPIED:F1:A TO /A/B
```

This command copies the Series II file :Fl:A to a file of the Series-IV called /A/B. A message verifying the copy operation appears on the console of the Series-IV.

3.
```
FPORT UP /A/B TO :F1:A
```

```
COPIED /A/B TO :F1:A
```

This command copies the Series-IV file /A/B to the Series II file :F1:A. A message verifying the copy operation appears on the console of the Series-IV.

4.
```
FPORT DOWN :F1:*.* TO /A UPDATE EXIT
```

```
COPY :F1:*.* TO /A UPDATE EXIT

COPY :F1:JUNK TO /A/JUNK ?
        Y<cr>

COPIED :F1:JUNK TO /A/JUNK

COPY:F1:JUNK.LST TO /A/JUNK.LST ?
        Y<cr>

COPIED :F1:JUNK.LST TO /A/JUNK.LST
```

In this example, the ISIS wildcard pathname *.* is used. It specifies that all files in the Series II directory file (:F1:) are to be copied to the iNDX directory (/A). The UPDATE option suppresses the querying and writes over existing files. After each file in the directory has been copied, a verification message appears. The EXIT option specifies that after the last file in the directory has been copied, the Series-II system should be returned to independent operation.

5.
```
FPORT UP /A/* TO :F1: QUERY
```

```
COPY /A/JUNK TO :F1: JUNK ?
        Y<cr>

COPIED /A/JUNK TO :F1: JUNK

COPY /A/JUNK.LST TO :F1:JUNK.LST ?
        Y<cr>

COPIED /A/JUNK.LST TO :F1: JUNK.LST
```

In this example, the iNDX wildcard character (*) is used to specify that all files within directory /A are to be copied to the ISIS directory :F1:. The QUERY option specifies that you should be queried before each copy operation.

6.
```
FPORT EXIT
```

This command releases the Series-II system from its slave status and returns it to independent operation.

Possible Error Messages

FILE DOES NOT EXIST
PATHNAME = *pathname*

INCORRECT FILE TYPE
PATHNAME = *pathname*
DIRECTORY FILE EXPECTED

FILE ALREADY EXISTS
PATHNAME = *pathname*

INSUFFICIENT ACCESS RIGHTS
PATHNAME = *pathname*
ADD ENTRY ACCESS TO DIRECTORY OR DELETE ACCESS TO EXISTING FILE REQUIRED.

INSUFFICIENT ACCESS RIGHTS
PATHNAME = *pathname*
READ ACCESS REQUIRED

MASS STORAGE EXCEEDED
PATHNAME = *pathname*

PATH COMPONENT NOT A DIRECTORY FILE
PATHNAME = *pathname*

INVALID WILD CARD PATHNAME
PATHNAME = *pathname*

ILLEGAL COMMAND SYNTAX

DISALLOWED USE OF A WILDCARD CHARACTER IN PATHNAME
PATHNAME = *pathname*

DEVICE TIMEOUT ERROR

Additional Notes

Program Restrictions:

1. FPORT must be used as a foreground job only.
2. FPORT does not support output devices :CO: and :BB: as valid pathnames.

Hardware Restrictions:

FPORT requires that a standard CRT (or "system-to-system") cable (pin 2-to-pin 2, pin 3-to-pin 3, no reversal) be connected between the J1 CH1/TTY connector on the back of the Series II and Serial Channel 2 on the back of the Series IV. The IEU must be in slot J9.

Error Handling:

The error handling is similar to and consistent with that defined for the copy utility program.

1. Syntax Error: Syntax is checked before operations begin. Invalid syntax causes a fatal error; an error message will be displayed before the command aborts. You must re-enter the command from the beginning.
2. Timeout Error: Whenever either system (Series II or Series IV) is expecting data but does not receive data, a timeout error occurs. Timeout errors, fatal errors, abort operation on both systems (Series II and Series IV).

ICOPY

Syntax

```
      ( READ   ISIS-source-pathname TO iNDX-destination-pathname ) ( QUERY  )
ICOPY {                                                          } { UPDATE }
      ( WRITE  iNDX-source-pathname TO destination-pathname      ) (        )
```

where

ISIS-source-pathname	is an ISIS filename or an ISIS wildcard filename as specified in the *ISIS-II User's Guide*, 9800306. The *ISIS-source-pathname* may optionally be preceded by a disk-directory.
iNDX-destination-pathname	is a valid iNDX directory file or iNDX pathname.
iNDX-source-pathname	is a valid iNDX pathname or wildcard pathname.
destination pathname	is a disk-directory, an ISIS filename, or an ISIS filename preceded by a disk-directory.
disk directory	may be :FO:, :F1:, :F2, or :F3: for Floppy disks or :F6, :F7, F8:, or :F9: for Hard disks.
QUERY and UPDATE	are options that determine if querying is to occur before files are copied.

Description

The ICOPY command has two forms: READ and WRITE. READ allows you to copy ISIS files to a specified location in the iNDX file structure. WRITE allows you to copy iNDX files to an ISIS file and, optionally, to an ISIS file pointed at by a disk directory.

Copying ISIS Files to iNDX

The READ function of the ICOPY command copies files from an ISIS source file to a destination file within the iNDX file hierarchy. If you specify an ISIS wildcard filename as the source file, the iNDX destination file must be an iNDX directory file. If the source file is not an ISIS wildcard file, the destination may be either an iNDX data file or a directory file.

If the iNDX destination file specified already exists and is a directory file, a file of the same name as the ISIS source file is created and placed in the iNDX directory file if you have ADD-entry access to the directory file. An error message is returned if the filename so constructed exceeds the iNDX limit of 127 characters.

If the iNDX destination file specified already exists and is a data file, you will be queried to see whether you want to destroy the existing file by writing over it. If you want to write over the existing file, and if you have delete access rights to the existing file, the existing file will be deleted and the new data file will be created. You may suppress the querying by specifying the UPDATE option in the command line.

If the iNDX destination file specified does not exist, a new file with the same name as the ISIS source file will be created within the parent directory if you have ADD-entry access to the parent directory.

Copying a File From iNDX to ISIS

The WRITE function of the ICOPY command allows you to copy an iNDX source file to an ISIS destination file. If you specify an iNDX wildcard pathname as the iNDX source file, the ISIS destination file must be disk-directory. If the source file is not an iNDX wildcard pathname, the ISIS destination file may be an ISIS file, a disk-directory, or an ISIS file pointed at by a disk-directory.

If the ISIS destination file is a disk-directory, a file of the same name as the iNDX source file will be constructed. If the filename so constructed is an invalid ISIS filename, a pathname syntax error message will be returned.

If the ISIS destination pathname already exists, you will be queried to see whether you want to delete the existing ISIS file by writing over it. If you reply by entering yes, the existing file will be deleted and the new file will be written to that filename unless the write and the format attributes for the existing file have been set. You may disable the querying by specifying the UPDATE option in the command line.

If the ISIS destination file does not yet exist, a new file will be created within the ISIS directory.

If you select a hard disk as the disk-directory, ICOPY will read the disk-directory to determine if the disk is an iNDX disk or an ISIS disk. If the disk is an iNDX disk, all files copied to it will be Public files (i.e., their Public attribute, but no other attribute, will be set).

Regardless of the hardware configuration, ICOPY always assumes the disk-directory configuration to be the iNDX configuration, as shown below:

:FO: to :F3:	represent Floppy drives 0 to 3, respectively, with :FO: as the default.
:F6:	represents the first Hard disk fixed platter.
:F7:	represents the first Hard disk removable platter.
:F8:	represents the second Hard disk fixed platter.
:F9:	represents the second Hard disk removable platter.

Physical Interaction

Physical interaction is required to specify the type of ICOPY operation to be performed. The command begins to execute as soon as you terminate the command line by entering <cr>. ICOPY then returns the following sign-on message to the console screen:

```
operating-system name ICOPY, Vx.y
```

ICOPY then checks all pathnames for valid syntax. If any syntax error is detected, the command aborts and you must re-enter the command from the beginning.

If the destination file already exists and is a data file, and you have not selected the UPDATE option to disable the querying process, the following query will appear on the screen:

```
FILE ALREADY EXISTS
PATHNAME = iNDX-pathname
DELETE EXISTING FILE ?
```

If you reply by entering a Y (or y) followed by <cr>, the command will continue to execute and the existing file will be deleted by being written over (provided you have delete access to the existing file). Any input other than Y (or y) causes the command either to go on to the next file to be copied (it the source file is a wildcard pathname) or to terminate. Since your responses to the queries are line edited, each response must be followed by <cr>. The querying process can be eliminated by specifying the UPDATE option in the command line.

You may specify the QUERY option with any invocation of the ICOPY command. Using this option causes the following query to appear before each file is copied:

COPY *source-pathname* TO *destination-pathname* ?

If you reply by entering Y (or y) followed by <cr>, the file will be copied. Any response other than Y (or Y) causes the command to either go on to the next file to be copied (if the source file is a wildcard pathname) or to terminate. Since your responses are line edited, they must be followed by <cr>.

If you are in Submit Mode and you use the QUERY option, the following error message will result:

```
QUERY OPTION NOT USEABLE IN SUBMIT MODE.
```

Each successful copy operation produces the following verfication message:

COPIED *source-pathname* TO *destination-pathname*

Additional Notes

Program Restrictions:

1. IEU must be present at the top of the user memory.
2. The hard disk controller is configured differently for ICOPY than it is for the Series-IV Operating System. Both programs cannot access the hard disk at the same time.
3. ICOPY can be invoked only from a foreground job.
4. **ICOPY will copy only Public files from any NDS-II files.**
5. ICOPY does not support single density disks.
6. In both the Read and Write functions of ICOPY, physical output devices such as :CO:, :BB:, :TO:, :LP:, etc., are not supported.
7. Wildcard pathnames used as destination pathnames are not supported.

Hardware Requirements

COPY requires that the Series-II hardware controller be installed in the Series-IV chassis. Before installing the controller boards in the Series IV, do the following:

A. For Double Density Floppy Disk Controller:
   1. Set the switches on the CHANNEL board to 78H port address as in ISIS.
   2. Set the rotary interrupt switch on the board marked INTERFACE completely in the clockwise direction—turning it past interrupt number 8. From 3
   3. Plug the boards into the Series-IV chassis. The board marked INTERFACE must be slot J2, J3, J5 or J7 (where J10 is the slot closest to the CRT and J1 the farthest).

B. For Hard Disk Controller:
   1. Set the switches on the board marked CHANNEL to 70H port address.
   2. Set the rotary interrupt switch on the CHANNEL board completely in the clockwise direction, turning it past interrupt number 8.

3. Set the switches on the board marked INTERFACE as follows:

   SW1—set all switches to the OFF position.
   SW2—set all switches to the OFF position.

4. Plug the board into the Series IV. The board marked CHANNEL must be in slot J2, J3, or J5 (where J10 is the slot closest to the CRT and J1 the farthest).

5. **In the NRM, plug the board marked CHANNEL into slot 12 and the marked INTERFACE into slot 13.**

Error Handling

Error handling is the same as that defined for the Copy command.

Syntax Errors—Syntax is checked before any operations begin. Invalid syntax is a fatal error; an error message will be displayed before the command aborts. You must then re-enter the command from the beginning.

Disk Errors—Disk errors are fatal errors resulting from unsuccessful disk I/O operations on the controller boards. Such errors result in an error message; the command aborts.

PDSCOPY

Syntax

```
PDSCOPY { READ  (disk-directory) PDS-source TO iNDX-destination } { QUERY  }
        { WRITE iNDX-source TO PDS-destination                  } { UPDATE }
```

where

PDS-source	is a valid PDS•filename or wildcard filename (that can optionally be preceded by a disk-directory).
iNDX-destination	is a valid iNDX directory file or pathname.
iNDX-source	is a valid iNDX pathname or wildcard pathname.
destination	is a valid PDS pathname (optionally preceded by a disk-directory).
QUERY	is an option that produces a user query before each PDSCOPY operation.
UPDATE	is an option that disables the automatic querying of PDSCOPY.

Description

PDSCOPY allows you to copy PDS files to a specified location within the iNDX file structure (READ) and to copy iNDX from the iNDX file structure to PDS files (WRITE).

Copying PDS Files to iNDX Files

READ allows you to copy a PDS file to a specified location within the iNDX hierarchy. If you enter a PDS wildcard as the source, you must designate an iNDX directory file as the destination file. If you do not designate a wildcard as the source, you may specify either a data or a directory file as the destination.

If the iNDX destination file exists and is a directory file, a file with the same name as the PDS source file will be created within the iNDX directory. If the pathname constructed for the destination file exceeds the iNDX limit of 127 characters, an error will be returned. The Read operation executes only if you have ADD-entry access to the destination directory.

If the destination file exists and is a data file, you will be queried before the delete takes places. Specifying the UPDATE option disables the querying. If the destination does not exist, a new file having the specified name will be created within the specified iNDX parent directory. You must have ADD-entry access to the parent directory to execute this command.

Copying iNDX Files to PDS Files

WRITE allows you to copy an iNDX file to a PDS file. If you designate an iNDX wildcard pathname as the source, the PDS destination must be a disk-directory; otherwise, the source may be either a PDS filename or a disk-directory.

If the destination is a disk-directory, a file with the same name as the source file will be constructed as the destination. If the pathname constructed is not a valid PDS pathname, an error will be returned.

If the destination pathname exists, you will be queried prior to deleting the existing file. Deletion will take place only if you request it and the Write and Format attributes of the existing file have not already been set. If you specify the UPDATE option, the querying will not take place before the deletion. If the file does not exist, a new file will be created within the PDS directory.

Regardless of your specific hardware configuration, the disk directory configuration is always taken by PDSCOPY to be one of the following:

:F0:—flexible disk drive 0 (the default condition)
:F1:—flexible disk drive 1

User Interaction

After you have entered the command line for PDSCOPY, the following message appears on the console:

```
operating system name PDSCOPY, Vx.y
```

PDSCOPY then performs a syntax check of the filenames entered. If any errors are detected, an error message will be returned and the command will abort. You must re-enter the command line with the correct filenames. If the filenames are correct, the command proceeds as described in the following paragraphs.

If the destination file already exists and is a data file (and you have not specified the UPDATE option—thus disabling querying), the following query will appear:

```
FILE ALREADY EXISTS
PATHNAME = destination pathname
DELETE EXISTING FILE ?
```

If you respond by entering a Y (or y) followed by <cr>, the command will proceed and the existing file will be deleted if you have access to the file. Any input other than Y (or y) terminates the command and no files are copied. If a wildcard file was the source, the command will then query you for the next file. If no more files exist, the command proceeds. Note that as the input for the PDSCOPY is line-edited, you must enter a line terminator at the close of the command lines. If you explicitly specify the QUERY option, the following query will appear before each copy takes place:

```
COPY source pathname TO destination pathname ?
```

If you respond by entering a Y (or y) followed by <cr>, PDSCOPY executes. Any response other than Y (or y) terminates the command. If the source was a wildcard, the command will query you for the next file. If no more files exist or the source was not a wildcard, the command will be exited. As the input is line-edited, you must end each command line with a line terminator.

In Submit Mode, the QUERY option causes the following message:

```
QUERY OPTION NOT USEABLE IN THE SUBMIT MODE.
```

Every successful copy causes the following confirmation message to appear:

```
COPIED source pathname TO destination pathname
```

Additional Notes

1. PDSCOPY can be invoked only from a foreground job.
2. PDSCOPY does not support the copying of spare files.

3. Wildcard pathnames cannot appear as the destination file.
4. For both READ and WRITE, the devices :CO:, :BB:, :TO:, :LP:, etc., are not supported.
5. Upon insertion of a PDS diskette into a Series IV, the iNDX Operating System attempts to mount the PDS diskette. This attempt fails, and a NOT SUPPORTED exception is returned. This message should be disregarded.
6. The possible errors are:

   Syntax Error: syntax is checked before operations begin. Invalid syntax is a fatal error and is displayed before aborting. You are required to re-enter the command from the beginning.

   Disk Error: fatal error. The program performed an unsuccessful Disk I/O operation on the controller boards. After this message is displayed, PDSCOPY is aborted.

VERIFY

Syntax

`VERIFY` *device-name* `[FIX]`

where

device-name	is: FL0 or FL1 for 5¼″ Flexible Disks HD0-HD3 for Hard Disks WM0 for the integrated 5¼″-Winchester Disk WD0-WD3 for a 35-Megabyte Priam Disk
FIX	is an option specifying that an appropriate action be taken, and an error message provided, if an error is detected.

Description

The Verify command verifies the volume (i.e., the disk) specified by checking:

1. Predefined fields in the System files, including file names.
2. The hierarchical file structure.
3. The integrity of the file system bitmaps.
4. The file contents of any bad blocks.
5. If deleted files have actually been deleted.
6. If bad blocks have been removed from usable disk space.
7. If allocated disk blocks are owned by more than one file.
8. If all allocated disk blocks are owned by a file.

The FIX option, if specified, checks for the following disk errors and corrects them as follows:

1. Defective predefined system files are corrected.
2. Files not traceable to a parent directory are deleted.
3. Files containing a bad block are deleted upon your directions (see "User Interaction" below).
4. Files that were incompletely deleted are deleted upon your instructions (see "User Interaction" below).
5. Bitmaps found in error are corrected.
6. Erroneous file allocation flags are corrected.
7. Bad blocks are marked as being allocated and thus are removed from the file system.
8. Disk blocks that are marked as used but are not owned by any file are recovered for (re)allocation.
9. Names of system files will be changed from old formatted names to new formatted names if a disk was originally formatted with an older version of the format command.

User Interaction

User interaction is necessary to Verify and Fix disks. After you have entered the command line for the Verify command, the following message appears on the console:

operating system `VERFIY` *Vx.y*

where

x.y	designates the current version of the software.

If a syntax error is detected in the command line, an error message will appear and you must re-enter the entire command. If no syntax error is detected, the following tests are performed in the sequence shown. For each test, one of the following messages appears, indicating the results of that test. The test result messages are:

`VERIFICATION PASSED`	no errors of this type were found.
`VERFICATION FAILED`	an error of this type was found.
`VERFICATION FAILED/FIXED`	an error of this type was found as you specified the FIX option; appropriate action was taken to correct the error.

Following are the Verification Tests. They appear in the order in which they are performed.

TEST #0 Verifying Formatted File Names

This test checks the predefined system filenames, comparing them against their expected contents. Discrepancies are reported as errors.

TEST #1 Verifying Predefined Fields in System Files

This test checks the predefined fields in the system files, comparing them against their expected contents. Discrepancies are reported as errors.

TEST #2 Verifying the Hierarchical File Structure

This test reads in all allocated files and determines if they are Directory files. All Directory files are read and all of their member files are marked as belonging to that parent directory. The contents of the parent directory are then checked against the roster of marked files. If an unmarked file is found, it is reported as a file without a parent directory.

TEST #3 Verifying File System Bitmaps

This test consists of five steps. (1) The bitmap that monitors file allocation is compared against the field in the file that marks whether that file has been allocated. A mismatch is reported as an error. (2) All allocated files are checked for type. An allocated file must be a System file, a Directory file, or a Data file. (3) A check is made to see if all files in the process of deletion have been completely deleted. (4) Files containing bad blocks are checked to see if they are unusable. Errors detected by the second, third and fourth steps result in the following error message:

```
PATHNAME is fully-qualified pathname
Delete (DIRECTORY OR DATA) file?
```

If you then respond by entering Y (or y) followed by <cr>, the file names will be deleted. Any response other than Y (or y) does not delete that file. Note that the verify command ignores file protection; thus, you may delete files that are not your own. (5) The final step is the comparison of the bad block bitmap against the free space bitmap. All disk blocks that are marked as bad in the bad block bitmap must be marked as used in the free space bitmap, thus preventing their future use.

TEST #4 Verifying Used Disk Block Allocation

This test consists of two steps. (1) All allocated files are read and all disk blocks belonging to each file are marked as such. If a block is encountered that had been previously marked to another file, the given block is owned by more than one file and an error message is displayed. (2) All used blocks are checked to see if they were marked in step 1. If a block was not marked in step 1, the block is being incorrectly marked as used but actually does not have an owner and therefore an error message is displayed.

When all four tests have been completed, one of the following messages is displayed:

```
DISK VERIFIES
```

No errors were detected so the disk is good.

```
DISK DID NOT VERIFY
```

At least one error was detected and so the disk is bad.

```
BAD DISK FIXED
```

Errors have been detected and some of the errors have been corrected.

Error Messages

Following is a list and explanation of all the error messages obtained with the Verify command.

ERR: SYSTEM FILE NAME ERROR
This error, returned by Test #0, indicates a mismatch between the existing name of a system file and the actual name found.

ERR: FILE(S) NOT TRACED BACK TO A PARENT DIRECTORY
This error, returned by TEST #2, indicates that a file cannot be traced to a valid parent directory.

ERR: FILE ALLOCATE FLAG FILE ALLOCATE BITMAP
This error, returned by TEST #3, indicates a conflict between the flag in a file indicating whether that file has been allocated and the corresponding flag in the File Allocate Bitmap.

ERR: INVALID FILE TYPE
This error, returned by TESTS #1 and #3, indicates that the type of a given file does not agree with the expected type.

ERR: FILE CONTAINS A BAD BLOCK
This error, returned by TEST #3, indicates that a bad block has been detected in the file whose name is given.

ERR: DELETED FILE ENCOUNTERED
This error, returned by TEST #3, indicates that a file was not completely deleted due to such circumstances as sudden device dismount during a delete or truncate, or user-reset of the system during a delete or truncation.

ERR: BAD BLOCK(S) NOT REMOVED FROM USABLE DISK SPACE
This error, returned by TEST #3, indicates that a bad block has been found to be marked as free in the free space bitmap. All bad blocks should be marked as allocated or used.

ERR:	DISK BLOCK(S) ALLOCATED TO MULTIPLE FILES This error, returned by TEST #4, indicates that a used block is marked as belonging to two different owners, which is illegal.
ERR:	DIRECT BLOCK COUNT < > INDIRECT BLOCK COUNT This error, returned by TEST #4, indicates that the total number or count of disk blocks used by a file is not equal to the count made by the Verify command. This is a system I/O error.
ERR:	DISK BLOCK(S) NOT ALLOCATED TO ANY FILE This error, returned by TEST #4, indicates that a disk block is marked as being used but is not owned by any file. This is illegal.

Fix Messages

The following messages are displayed when an error is detected and you have specified the FIX option.

fix:	system file content corrected This fix message, returned by TEST #1, is displayed when defective system file contents have been corrected.
fix:	bad file(s) deleted This fix message, returned by TESTS #2 and #3, is displayed when the Verify command has determined that a particular file is bad and cannot be accessed. Bad files are defined as: (1) those not traceable to a parent directory, and (2) files that contain a bad block. Files with a bad block are deleted only if you explicitly request such action.
fix:	bad file flag corrected This fix message, returned by TEST #3, is displayed when the Verify command has determined that the allocation flag is bad. Verify changes the flag to its correct status.
fix:	bad file bitmap corrected This fix message is displayed when the Verify command has determined that the bitmap monitoring the file allocation is bad. The bitmap is changed to correct status.
fix:	bad blocks removed from usable disk space This fix message, returned by TEST #3, is displayed when bad blocks are detected in the bitmap that monitors whether a disk block is used or available.
fix:	lost blocks recovered This fix message, returned by TEST #4, is displayed when the Verify command has marked lost blocks (i.e., disk blocks previously marked as used but not owned by an file) as available or free.

Examples

1. `VERIFY FL1`

   This example Verifies 5¼″ flexible drive 1.

2. `VERIFY HD0 FIX`

   This example Verifies the hard disk drive 0 and fixes detected errors.

Additional Notes

1. The use of the Verify command is restricted to foreground jobs only.
2. The Verify command dismounts the device, making it unable to verify a system drive.
3. The Verify command does not free blocks that are multiply allocated.
4. The Verify command will not run concurrently with a background job.

CHAPTER 5
USING THE SERIES-IV IN THE NETWORK

The Series-IV workstation is used in the same manner whether your workstation is part of the NDS-II Network or is a standalone development system. The remainder of this manual describes what you need to know to interface with your Series IV. This chapter gives an overview of the NDS-II Network Development System and describes the additional operations needed to use the Network.

The NDS-II Network

The NDS-II, together with the optional hardware and software packages available from Intel, is a complete local network development solution for microcomputer applications.

The Intel hardware, the operating system software, and the communications subsystem based on the Intel Network Architecture (iNA) offer a distributed processing development environment. This environment allows all network users at one development system workstation to use the processing resources of other Intellec Development System workstations. The workstations can connect logically to the Network, where all users share concurrent access to a public file system on shared mass storage devices, a spooled line printer, and distributed job control facilities.

Physical Description

The NDS-II hardware has five main parts:

1. A Network Resource Manager (NRM) with a CRT terminal attached
2. Ethernet Communication boards in the NRM and the workstations
3. An Intellink Communications Module and cables to connect the NRM and the workstations to the Intellink or Ethernet cable
4. A line printer (optional)
5. Intellec Development System workstations (up to 16)

Functional Description

NDS-II offers distributed processing with local workstation resources and remote network resources. The Network also acts as a host for software tools (ASM, Fortran, Pascal, and PL/M), instrumentation tools (ICE® In-Circuit Emulators), and program management tools (hierarchical file structure, MAKE and iSVCS).

Each workstation in the Network shares concurrent access to the shared file system, the remote execution job queues, and the printer queue.

The distributed network file system offers several advantages: large mass storage capability, peripheral device sharing, enabling higher performance at a lower cost per workstation, file sharing among project members and with other projects, distributed hierarchical file system, and archival facility for files stored on the shared disks.

Remote job execution allows a workstation user to execute a batch job on another NDS-II workstation. Remote job execution offers higher throughput and greater efficiency because more than one processor on the Network can be used by a single user.

Remote Jobs

The Network Job Control recognizes that the network has three types of stations: the NRM, private workstations, and import workstations. The NRM is the nerve center of the network job control; it maintains all the state information of remote jobs and the status of the workstations. A private workstation is one that can send jobs to the NRM and have them executed by other workstations in the network; but, it does not accept jobs from the NRM. When a workstation is first powered up, it is a private station. An import workstation is one that can accept jobs from the NRM. A private workstation can be turned into an import station with the IMPORT command. Once a station becomes an import station, it remains that way until CONTROL-C is pressed from its keyboard or until an EXIT_IMPORT message is received from the NRM. If the import workstation is executing a remote job when CONTROL-C is pressed, the import station continues executing that job until the job is finished. Only when the job is finished does it turn itself into a private workstation again. A Series-IV station that supports both foreground and background can import into either foreground, background, or both; thus, a physical station may appear as two stations to the Network Job Control.

The network consists of heterogeneous workstations; thus, some kind of mechanism is needed to match the job with the type of workstation it needs to execute. The network queue is one way to solve this problem. The Network Job Control maintains a number of queues at the NRM. Each queue has a number of import workstations declared as the queue servers. Queues are created statically with the QUEUE command; the system does not have any predefined queues. You are responsible for determining a convention to establish the mapping between the queue names and the types of jobs the servers of that queue can handle. One example is to use SERIESII as the queue name for the Series-II job queue and SERIESIV as the queue name for the Series-IV job queue. The system does not and cannot check that a job is sent to a queue which can handle that job.

When a private station is turned into an import workstation, you must specify from which queues that station is to receive jobs. The import station is added to the list of servers of those queues. When you send a job to the NRM, you must specify in which queue you want the job placed. The system guarantees that the job is executed only by a server of the specified queue. Although you cannot guarantee exactly which station (of those serving that queue) will execute that remote job, a properly designed queue-naming convention insures that only a workstation of the right type will execute the job.

A remote job is similar to a background job. The major difference between these two is a remote job is created by the EXPORT command, executed on a station other than the one with which you are interacting, and you have to specify the name of the queue where the job awaits execution. The remote job may or may not be started immediately depending upon the availability of a server station.

Any (external) resources or files required by an EXPORT command must be made available at the importing workstation that will execute the job; local resources and files are not transported to the importing station as part of the EXPORT command.

I/O Differences

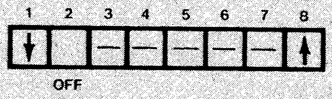
You can queue printout with the Network, allowing you to send your output to the spool printer and to use your workstation to initiate another job. Note, however, that the device name for the printer in this case is :SP:, not :LP:, as it is in the standalone mode. :LP: designates the local line printer, not the network spool printer. Any output from workstations in the network mode should be sent to the spool printer (:SP:).

Initiating Workstation Operation

The following procedure assumes the System Generation procedure has been implemented and the NDS-II network configuration has been verified. (See the *NDS-II System Generation Instructions*, 122003.)

To initiate workstation operation:

1. Check that the Network Resource Manager is operative. See the *NDS-II Network Resource Manager Operating Instructions*, 122883.
2. Verify the configuration switches are set as illustrated below.



3. Power up the Series IV by pressing on the breaker switch located at the left rear of the terminal chassis.
4. Verify the sign-on message is displayed. If it is not, press the RESET switch (see figure 1-3) to boot the operating system.
5. Verify the diagnostics test passed. If it did not, refer to the *Intellec Series IV Installation and Checkout Manual*, 121757.
6. Enter your assigned user name.
7. Enter your password.
8. Turn on any peripherals such as a Winchester disk drive or a printer. When using 740 Hard Disk Drives, press the STOP/START button and wait for the READY light before proceeding.

The system is now ready to accept commands. Use the facilities detailed in the *Series-IV Overview*, 121752, to assist you in learning to interface with your system as you design and develop your projects on the Series IV.

Terminating Operation

Terminate User Session

To terminate the user session and to allow another user to start a session:

1. Log off by entering the command LOGOFF or by pressing the appropriate soft key.
2. The new user may now log on by entering his user name and password.

The system is now ready to accept commands from the new user.

Power Down Development System

To shut off your workstation:

1. Log off by entering the command LOGOFF or by pressing the appropriate soft key.
2. Turn off any peripheral devices.
3. If you are using a flexible disk(s), wait for the drive indicator light to go off (see figure 1-2). Release the drive door latch(es) and remove the flexible disk(s).
4. Move the rocker switch located at the left rear of the terminal chassis to the off position.

Remote Job Commands

The remote job commands are:

CANCEL
EXPORT
IMPORT
QUEUE
SYSTAT

They are described in the following pages.

CANCEL

Syntax

```
CANCEL { BACKGROUND
       { REMOTE       queue  { (jobname)  (#jobnumber) } }
```

where

queue	is the queue where the remote job is queued for execution.
jobname	is the final component name of the remote job to be cancelled.
jobnumber	is the assigned value of the remote job (which can be displayed via the SYSTAT command).

Description

The Cancel command is used to cancel a background (see the Cancel command in Chapter 4) or remote job. If you want to abort a remote job, you must enter the *jobname* or *jobnumber* that will be aborted. *jobname* is the final component name used in the Export command that placed the job on the queue. You can only cancel jobs you have sent to the queue.

If *jobname* is selected and multiple instances of *jobname* are in the queue, the first *jobname* encountered is deleted (this may not be the first one queued). To avoid this confusion, use the unique name *jobnumber*.

EXPORT

Syntax

EXPORT *pathname* [*parameters*] TO *queue* $\left[\begin{Bmatrix}\text{LOG}\\\text{NOLOG}\end{Bmatrix}\right]$

where

pathname	is a valid pathname.
parameters	is a list of up to ten parameters.
queue	is the queue to which the job is to be sent.
LOG, NOLOG	specifies whether a log is to be kept on a mass storage device of any console activity.

Description

The Export command allows a command file composed at one workstation to be executed on a remote workstation. The command file specified must be on a public volume so that a Public workstation can have access to it. The Network Resource Manager (NRM) maintains a queue of jobs to be executed at remote public workstations. The desired queue is specified in the command line. If the queue does not exist, an exception message is displayed and the job is not queued.

LOG, NOLOG determines whether a log is to be maintained on a mass storage device of console activity. LOG is the default condition.

The Background, Remote, and Submit commands all have similar structures. These similar structures permit you to execute any given command file with any of these commands.

If your command file name has no extension, the system appends the extension .CSD to the command file. If your filename ends with a period, the system will use the filename you have given (with the period truncated). If your filename has an extension, the system uses that filename without modifying it.

The optional parameters specified in the command line are the actual parameters to be substituted for the formal parameters embedded within the command file. A maximum of ten actual parameters can be specified in the command line. Placing formal parameters in the command file allows you to call the same command file for varying sets of actual parameters. When no parameters are specified in the command line, the specified command file is directly executed.

When parameters are specified, a second command file is generated. The actual parameters in the second file are sequentially substituted for the formal parameters. Thus, (reading from left-to-right) if you enter the actual parameters ASM, PLM, and BLT, ASM replaces the formal parameter %0, PLM replaces %1, and BLT replaces %2. The command file resulting from the parameter substitution is placed in the same directory as the generating command file. The final path component is derived from the final component of the generating file by:

- The final component after the third character or before the first period (whichever comes first) of the generating file is truncated.
- A unique character string obtained when the system job manager and the extension .TMP are appended.

When the LOG option is specified, a log is routed to a mass storage device. When LOG is specified, the LOG file takes the extension .LOG. NOLOG specifies that no such log is to be kept. If neither option is explicitly entered, LOG is the default condition.

Examples

Refer to the examples given under the Submit command for a comprehensive illustration of the manner in which command files are generated.

IMPORT

Syntax

```
IMPORT FROM queue [,queue]... [TO BACKGROUND]
```

where

queue	is a character string up to 14 characters long which names the queue at the point where the job awaits execution. Up to 5 queues may be specified in one command.
TO BACKGROUND	is an option that will execute the imported job in background mode.

Description

The Import command must be invoked interactively from the foreground of the given workstation. You can invoke the Import command while logging on or any time after logging on.

The Import command declares the given workstation to be a Public Workstation. A Public Workstation is one that can receive jobs maintained by the Network Resource Manager (NRM). If you enter the name of a queue that does not exist at the NRM, an exception message is displayed. The queues are searched for jobs according to the order in which the queues were listed in the command line (left-to-right). If jobs are available in the queues, the importing station will begin processing them. The importing station starts by performing an implicit log on for you who created the first job taken off the Queue, then processes commands within the command file. At the end of the command file, the importing station logs you off and looks for another job from the queue to process.

All output messages from the remote job are displayed on the screen of the importing station and may be put on a log file of a mass storage device if the LOG option is specified in the Export command.

NOTE

IMPORTED jobs are implicitly logged on. Initialization options are identical to those specified in LOGON.

When a station becomes a Public Workstation, it becomes unavailable for any local processing. To return the station to private operation so it will stop accepting jobs from the queue, hold down the CONTROL key and press the C key (CONTROL-C).

Each public workstation can service up to five queues. Each network can contain up to ten queues.

QUEUE

Syntax

```
QUEUE
```

Description

The QUEUE command is used for management of the queues present at the NRM. For each queue, the QUEUE command displays the name, number of jobs outstanding, and the number of servers. After this information is displayed, you are prompted:

```
ADD DELETE LIST EXIT
```

The ADD option creates new queues. Ten queues can exist at one time. The DELETE option lets you delete a queue. The LIST option re-displays previously listed information. The EXIT option terminates use of the QUEUE command.

SYSTAT

Syntax

```
SYSTAT[[{QUEUE}(queuename [, ...])]][TO pathname] [EXPAND] [ALL]
        [{MYJOB}                  ]
```

where

queuename(s)	designates the name(s) of the queue(s) for which jobs are to be listed.
pathname	designates the file where the information will be listed.
QUEUE	displays information for all queues, or for only those queues explicitly listed after the QUEUE specifier. If this option is specified, the queuenames must be separated by commas.
MYJOB	parallels the QUEUE option but lists information about jobs belonging only to you.
EXPAND	specifies that complete information is displayed for each job. If EXPAND is not specified, condensed information will be displayed. Complete and condensed information are demonstrated below in examples 4 and 2, respectively.
ALL	displays appropriate information for all jobs in the specified queue(s). If ALL is not specified, information is displayed only for jobs that are waiting or executing.

Description

The QUEUE option specifies that jobs of all users from the specified queue list be displayed. The MYJOB option specifies that only your jobs are displayed. If no queue list is specified, the requested information is displayed for all queues. If you specify more than one queue, you must separate the queuenames by commas.

If you use the specifier ALL, information is provided for all jobs in the selected queues regardless of their status. If you do not specify ALL, information is provided only for those jobs that are executing or waiting.

Examples

1. SYSTAT

```
SYSTAT cusp version V1.0
QUEUE                 # OF JOBS            # OF IMPORT
 NAME                 WAITING              STATIONS
QUEUE2                   2                    0
QUEUE3                   1                    1
```

2. SYSTAT QUEUE

```
SYSTAT cusp version V1.0
JOB STATUS FOR:  QUEUE2
 JOB NAME            JOB #      OWNER      DATE        TIME       STATUS
 ABAK                1003         A        11/11/11    11:27:49   WAITING
 ABAK                1001         A        11/11/11    11:16:40   WAITING
JOB STATUS FOR:  QUEUE3
 JOB NAME            JOB #      OWNER      DATE        TIME       STATUS
 ABAK                2001         A        11/11/11    12:03:46   WAITING
```

3. SYSTAT QUEUE ALL

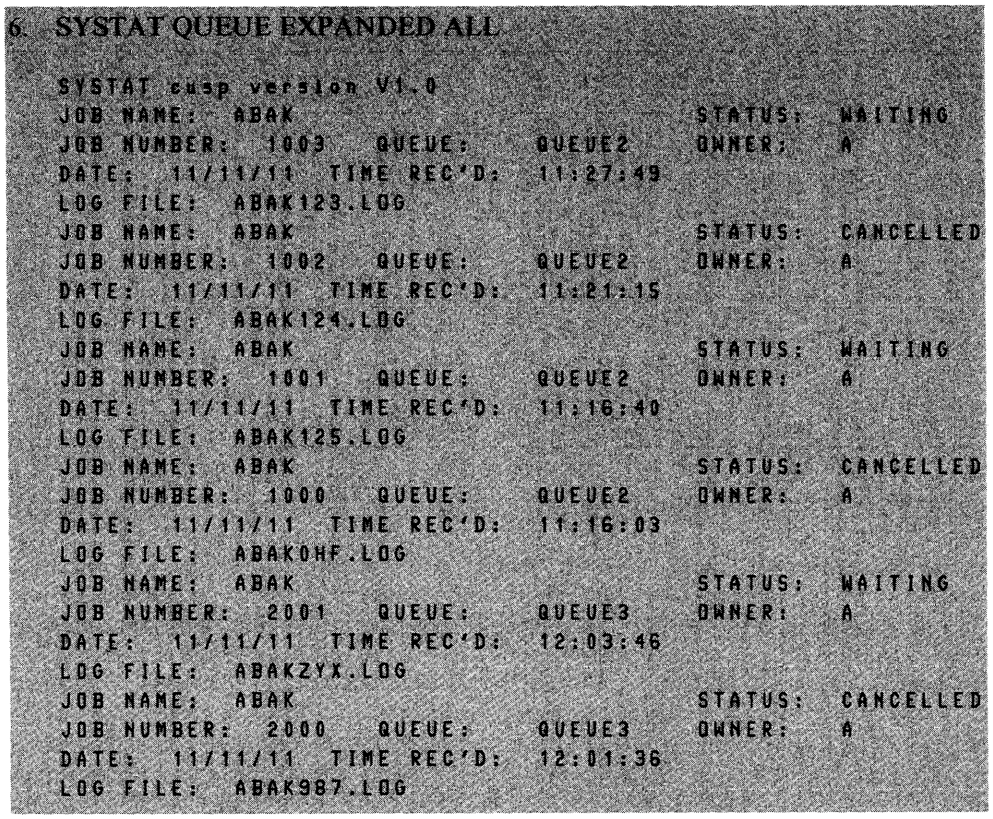
```
SYSTAT cusp version V1.0
JOB STATUS FOR:  QUEUE2
 JOB NAME           JOB #      OWNER     DATE       TIME       STATUS
 ABAK               1003         A       11/11/11   11:27:49   WAITING
 ABAK               1002         A       11/11/11   11:21:15   CANCELLED
 ABAK               1001         A       11/11/11   11:16:40   WAITING
 ABAK               1000         A       11/11/11   11:16:03   CANCELLED
JOB STATUS FOR:  QUEUE3
 JOB NAME           JOB #      OWNER     DATE       TIME       STATUS
 ABAK               2001         A       11/11/11   12:03:46   WAITING
 ABAK               2000         A       11/11/11   12:01:36   CANCELLED
```

4. SYSTAT QUEUE EXPANDED

```
SYSTAT cusp version V1.0
 JOB NAME:  ABAK                                STATUS:  WAITING
 JOB NUMBER:  1003    QUEUE:     QUEUE2         OWNER:   A
 DATE:  11/11/11  TIME REC'D:   11:27:49
 LOG FILE:  ABAK123.LOG
 JOB NAME:  ABAK                                STATUS:  WAITING
 JOB NUMBER:  1001    QUEUE:     QUEUE2         OWNER:   A
 DATE:  11/11/11  TIME REC'D:   11:16:40
 LOG FILE:  ABAK124.LOG
 JOB NAME:  ABAK                                STATUS:  WAITING
 JOB NUMBER:  2001     QUEUE:     QUEUE3        OWNER:   A
 DATE:  11/11/11  TIME REC'D:   12:03:46
 LOG FILE:  ABAK132.LOG
```

5. SYSTAT MYJOB

```
SYSTAT cusp version V1.0
JOB STATUS FOR:  QUEUE2
 JOB NAME           JOB #      OWNER     DATE       TIME       STATUS
 ABAK               1003         A       11/11/11   11:27:49   WAITING
 ABAK               1001         A       11/11/11   11:16:40   WAITING
JOB STATUS FOR:  QUEUE3
 JOB NAME           JOB #      OWNER     DATE       TIME       STATUS
 ABAK               2001         A       11/11/11   12:03:46   WAITING
```



6. SYSTAT QUEUE EXPANDED ALL

```
SYSTAT cusp version V1.0
JOB NAME:  ABAK                                 STATUS:  WAITING
JOB NUMBER:  1003   QUEUE:    QUEUE2    OWNER:   A
DATE:  11/11/11  TIME REC'D:  11:27:49
LOG FILE:  ABAK123.LOG
JOB NAME:  ABAK                                 STATUS:  CANCELLED
JOB NUMBER:  1002   QUEUE:    QUEUE2    OWNER:   A
DATE:  11/11/11  TIME REC'D:  11:21:15
LOG FILE:  ABAK124.LOG
JOB NAME:  ABAK                                 STATUS:  WAITING
JOB NUMBER:  1001   QUEUE:    QUEUE2    OWNER:   A
DATE:  11/11/11  TIME REC'D:  11:16:40
LOG FILE:  ABAK125.LOG
JOB NAME:  ABAK                                 STATUS:  CANCELLED
JOB NUMBER:  1000   QUEUE:    QUEUE2    OWNER:   A
DATE:  11/11/11  TIME REC'D:  11:16:03
LOG FILE:  ABAKOHF.LOG
JOB NAME:  ABAK                                 STATUS:  WAITING
JOB NUMBER:  2001   QUEUE:    QUEUE3    OWNER:   A
DATE:  11/11/11  TIME REC'D:  12:03:46
LOG FILE:  ABAKZYX.LOG
JOB NAME:  ABAK                                 STATUS:  CANCELLED
JOB NUMBER:  2000   QUEUE:    QUEUE3    OWNER:   A
DATE:  11/11/11  TIME REC'D:  12:01:36
LOG FILE:  ABAK987.LOG
```

CHAPTER 6 PROGRAMMING INTRODUCTION

Operating System Considerations

An operating system is a group of programs that provide the functional (as opposed to physical) environment in which your programs do their work. The group of programs facilitates efficient production and allocation of resources.

Needed Capabilities

An operating system is capable of managing the devices attached to the system hardware. These devices include the console input and output devices and auxiliary memory devices such as flexible or hard disk drives. The operating system also recognizes commands to invoke the execution of programs such as language translators or programs you develop.

Desirable Features

By managing overlays and error conditions, you can extend the range of functions (and recoveries) available to your programs. Overlays permit the design and use of a program larger than the memory size by partitioning the program into modules whose processing is mutually exclusive and whose size allows them to fit into the memory available. These modules are linked by calls that cause memory to be reused by the other sections of the program. Service routines designed to handle exceptions allow early detection and handling of unwanted conditions arising during execution.

Functions of the iNDX and ISIS-IV Operating Systems

The iNDX and ISIS-IV operating systems have the capabilities just described. They are also capable of command scanning and dynamic file or device manipulation (i.e., under program control during execution).

Command scanning allows your program to pick up options specified on the input line that invokes program execution, or to treat specially formatted files as if they were input from the console.

Dynamic file control during execution allows you to maintain a list of twelve files or devices that are used by your program (e.g., written or read); but, you can only use six at any one time. The ability to immediately access twice as many files and devices can make input/output operations more efficient.

The Series-IV operating system also provides memory management and an interactive symbolic debugging aid.

Memory management during execution allows you to allocate memory for specific processes as they arise and to free those blocks when they are no longer needed.

The debugging tool is called DEBUG-88. Its interactive language is similar to that of Intel's ICE-86 or ICE-88 emulators. Using DEBUG-86, you can insert breakpoints into your program, execute until some predefined condition is encountered, and halt—thereby allowing you to examine the state of processor registers or variables in your program.

DEBUG-88 is fully described in the *DEBUG-88 User's Guide*, 121758.

Program Development Cycle

The program development cycle begins with an idea and ends with a fully checked-out program that performs the desired work acceptably.

The idea evolves into a design and, ultimately, into program specifications. Specifications are split up into smaller groups of functions, each performed by a single module of code.

As work progresses, problems may arise—due, perphaps, to unforeseen gaps or complications within the base design or difficulties in implementing it. Modules may require expansion, modification, or integration with other modules. Some functions may merge or be abandoned, and parameter lists may change.

To minimize such changes (and the redesign, rework, and relearning resulting from them), you should remember the following guidelines:

1. Develop extremely clear and specific goals for the program. Write down the goals and have the designers, implementers, and users all agree on them.
2. Isolate every non-trivial function of the system or program into separate modules; even isolate difficult design decisions.
3. Write full and clearly understandable documentation for every module, including liberal comments in the code.
4. Write clear standards for implementating modules, including conventions for naming and passing parameters.

The closer you adhere to these guidelines, the farther you will get from unexpected, costly changes.

Modules defined through the above process then form the units of actual programming work which are separately specified at the detail level. They are coded, translated, and tested, both individually and in logical groups.

As these groups are tested and combined with other checked-out groups, the complete program approaches final integration. Using input data that reflects the ultimate usage as closely as possible, the final tests explore every major option defined by the original program specifications.

At each stage of individual and multi-module testing, the debugging functions provided in the operating system help to isolate the source of unexpected results. Under the Series-IV operating system, DEBUG-88 permits: use of symbolic names for debugging output; references to instructions by line number; access to the processor's registers and flags; and alternate execution modes with or without the use of breakpoints. Under the ISIS-IV operating system, the Monitor debugger or an In-Circuit Emulator provide similar functions.

Specific System Services for Each Target Environment

When you are developing a program to run on the Series IV, you must choose one of the two environments provided for program execution: the 8086/8088-based environment or the 8080/8085-based environment. Each has similar built-in facilities to aid you in the development and testing of your program products, including standard system services that can be called from your programs (e.g., I/O). These I/O routines free you from rewriting routines already embedded in the operating system and provide a standard interface for all modules or systems you develop. However, the interface for each operating system environment is unique; the calls and parameters differ.

The 8086/8088-Based Environment

For the 8086/8088-based environment, you develop the modules using the languages that run on the 8086/8088 and produce code that works on the 8086/8088 (for example, the resident FORTRAN-86, PASCAL-86, ASM-86 or PL/M-86 translators). (Earlier versions of these translators ran on the 8080/8085 chip but produced code to run on the 8086/8088 chip. In some cases, modules compiled or assembled with these prior versions of the translators can be used unchanged. Appendix D clarifies the circumstances in which this is workable.)

Over two dozen system service routines are available in the 8086/8088-based environment. These routines enable you to use the capabilities of the Series-IV operating system to manage the resources of the 8086/8088-based environment.

A conceptual introduction of expected parameters and results of routine execution appears early in Chapter 7. A full discussion of each parameter used also appears with the sample PL/M-86 declarations for these external procedures. The discussion of each routine ends with syntax examples and the list of exception conditions that can occur during the routine's execution. Brief, combined usage examples appear after these discussions.

Whenever you write a module that uses one of these service routines, you simply declare it as an external procedure. LINK86 then provides the correct address to the resident system program. You specify the library appropriate to the PL/M-86 model of segmentation you programmed for: SMALL.LIB, COMPAC.LIB, or LARGE.LIB.

After you link and locate groups of modules that will work together in your final system, you can test them alone or together. The DEBUG-88 feature described in the *DEBUG-88 User's Guide*, 121758, can help you isolate and correct defects as they become apparent.

The 8080/8085-Based Environment

For the 8080/8085-based environment under the ISIS-IV operating system, use the standard 8085-based versions of these same translators to build your modules. These modules will then run under ISIS-IV on the 8080/8085. In this environment, the modules may call upon ISIS routines for a variety of input/output services that already exist as part of the ISIS-IV facilities. Many of these routines operate similarly to those of the 8086/8088-based operating system; some, however, are unique to ISIS-IV.

Fourteen ISIS-IV routines and nine Monitor routines are available.

As under Series-IV, whenever you write a module that uses one of these service routines you simply declare it an external procedure. When the module is processed by LINK, the correct address to the resident system program is provided. The DEBUG feature of the Monitor can aid the program analysis and correction process. (Refer to the *ISIS-IV User's Guide*, 121880, for further discussion of the 8080/8085-based environment).

Built-in Service Routines

The parameters appropriate to each service routine include the address of a word (filled by the system) indicating whether the desired operation finished successfully. Usually, your call should be followed by code that tests this word—permitting error recovery, alternate processing, or exit, depending on the operation's results.

To help you understand the available service routines, the following three tables name all of the 8086/8088-based system routines:

1. Alphabetically (table 6-1)
2. In groups by function (table 6-2)
3. By sequence of use in a hypothetical program. Table 6-3 shows their nearest functional equivalent under ISIS-IV. (Some procedures used under the Monitor of ISIS-IV have no direct counterpart in the 8086/8088-based environment.)

All 8086/8088-based procedures begin with DQ$.

Table 6-1. Alphabetical List of Service Routines Available in the Series-IV Operating System

DQ$ALLOCATE	DQ$GET$SIZE
DQ$ATTACH	DQ$GET$SYSTEMS$ID
DQ$CHANGE$ACCESS	DQ$GET$TIME
DQ$CHANGE$EXTENSION	DQ$OPEN
DQ$CLOSE	DQ$OVERLAY
DQ$CREATE	DQ$READ
DQ$DECODE$EXCEPTION	DQ$RENAME
DQ$DECODE$TIME	DQ$RESERVE$IO$MEMORY
DQ$DELETE	DQ$SEEK
DQ$DETACH	DQ$SPECIAL
DQ$EXIT	DQ$SWITCH$BUFFER
DQ$FILE$INFO	DQ$TRAP$CC
DQ$FREE	DQ$TRAP$EXCEPTION
DQ$GET$ARGUMENT	DQ$TRUNCATE
DQ$GET$CONNECTION$STATUS	DQ$WRITE
DQ$GET$EXCEPTION$HANDLER	

Table 6-2. Service Routines by Functional Groups

Utility and Input Scanning
DQ$DECODE$TIME
DQ$GET$ARGUMENT
DQ$GET$SYSTEM$ID
DQ$GET$TIME
DQ$SWITCH$BUFFER
Memory Management
DQ$ALLOCATE
DQ$FREE
DQ$GET$SIZE
DQ$RESERVE$EXCEPTION

Table 6-2. Service Routines by Functional Groups (Cont'd.)

File Management
Program Connection and File Existence
DQ$ATTACH DQ$CREATE DQ$DELETE DQ$DETACH DQ$FILE$INFO DQGETCONNECTION$STATUS
Naming
DQ$CHANGE$ACCESS DQ$CHANGE$EXTENSION DQ$RENAME
Program Usage
DQ$CLOSE DQ$OPEN DQ$READ DQ$SEEK DQ$SPECIAL DQ$TRUNCATE DQ$WRITE
Program Control
DQ$EXIT DQ$OVERLAY
Exception Handling
DQ$DECODE$EXCEPTION DQGETEXCEPTION$HANDLER DQ$TRAP$CC DQ$TRAP$EXCEPTION

Table 6-3. Hypothetical Steps in Program Execution and Service Routines Relevant to Each Step

Steps	Service Routine Names	
	For Use in 8086/8088 Environment	For Use in 8080/8085 Environment*
1. Finds out date and system i.d. for logging/reporting purposes	DQ$DECODE$TIME DQGETTIME DQGETSYSTEM$ID	none
2. Allocates a memory work area for intermediate calculations	DQ$RESERVE$IO$MEMORY DQ$ALLOCATE DQGETSIZE	none
3. Determines whether console input is trans parent or line-edited	DQ$SPECIAL	none: console is always line-edited
4. Rescans the last command (at first, the one invoking this program)	DQGETARGUMENT DQ$SWITCH$BUFFER	RESCAN
5. Asks user to enter needed data or parameters at the console	DQ$WRITE DQ$READ	WRITE READ
6. Writes to a file	DQ$WRITE	WRITE

Table 6-3. Hypothetical Steps in Program Execution and Service Routines Relevant to Each Step (Cont'd.)

Steps	Service Routine Names	
	For Use in 8086/8088 Environment	For Use in 8080/8085 Environment*
7. Loads overlay to process next phase or user response	DQ$OVERLAY	LOAD
8. Checks to see if required files are on-line; gets status, including file pointer position	DQ$ATTACH/DQ$OPEN DQGETCONNECTION$STATUS	SEEK/OPEN
9. Creates files as needed for program reads/writes	DQ$CREATE/DQ$OPEN	OPEN
10. Opens the files for reads/writes	DQ$OPEN	OPEN
11. Reads file(s) or seeks to desired position in file	DQ$READ DQ$SEEK	READ SEEK
12. Calculates	user supplied	none
13. Frees memory work areas no longer needed	DQ$FREE	none
14. Closes and/or deletes files	DQ$CLOSE DQ$DELETE	CLOSE DELETE
15. Writes new or old file(s)	DQ$WRITE	WRITE
16. Renames certain files or changes extension on filename string; changes file protection attributes	DQ$RENAME DQ$CHANGE$EXTENSION	RENAME none
17. Change or check file access rights	DQ$CHANGE$ACCESS DQ$FILE$INFO	ATTRIB/GETATT
18. Obtains file device directory information	none	GETD
19. Truncates and/or closes files no longer needed	DQ$TRUNCATE DQ$CLOSE	CLOSE
20. Detaches files not currently needed	DQ$DETACH	none
21. Repeats as needed from number one above	none	none
22. Naturally, errors or exceptions or unwanted conditions can occur at each of the above steps. Thus, an implicit step after any of them is the detection/handling of such conditions.	DQ$TRAP$EXCEPTION DQ$DECODE$EXCEPTION DQ$TRAP$CC DQGETEXCEPTION$HANDLER	ERROR
23. Exits when job is complete or cannot continue	DQ$EXIT	EXIT

*Some names repeat because routine is multi-function.

CHAPTER 7
THE 8086/8088-BASED ENVIRONMENT

This chapter discusses each system service routine available in the 8086/8088-based environment of the Series IV. The routines provide a variety of capabilities to programs running on the Series IV. The routines do not, however, require user development and verification because they are part of the operating system.

Conceptual Considerations

The system service routines, which embody a variety of usage expectations, constitute a model of the way programs interact with files, the console, and each other. The expectations are directly reflected in the parameters you must supply when calling the routines. Following are some of the key concepts underlying the parameters.

Command Tail Arguments

An 8086/8088-based program such as PROGRM can be invoked by typing PROGRM. PROGRM may have options that can be specified on the invocation line. If so, the remainder of that line, including any continuation lines, is called a "command tail."

This command tail is accessible to PROGRM via the DQ$GET$ARGUMENT system service routine, which you call to get each option in the command tail. The first parameter of this call tells the system where to put the next option found, i.e., the address of the name you declared in PROGRM as the string to receive these options.

Successive calls to DQ$GET$ARGUMENT return successive options, each separated by some delimiting character such as a blank or a parenthesis. (Details for using this routine appear later in this chapter.) The concept of the command tail is basic to the discussion of that routine and can influence your program design.

Memory Management

Memory management routines monitor which memory areas are in use and which are free to be allocated to new uses.

Free space memory management is handled by the service routines DQ$ALLOCATE, DQ$FREE, and DQ$GETSIZE.

The Series IV does not support absolute object modules, but does support two types of relocatable object modules: position-independent-code (PIC), and load-time-locatable (LTL). Segment register changes do not occur in PIC modules. The code can work wherever it is ultimately loaded. LTL modules contain special records to resolve program references that do require segment register changes, e.g., an intersegment jump.

When a relocatable object module (PIC or LTL) is loaded, the lower limit of the free space pool is set before the load. Memory required to load the segments is then allocated from the initial free space pool by the free space manager.

A request for memory (i.e., invoking DQ$ALLOCATE) will return the lowest-addressed segment within the requesting job's region. When a segment is freed, it is automatically combined with adjacent free memory to form the largest contiguous area possible.

Connections

The operating system maintains a list of twelve devices or files your program can use during its execution, i.e., a list of "connections." A connection is a word, named by you, filled by the DQ$ATTACH or DQ$CREATE system service routines. (Only six connections may be open at once, although multiple opens of a single device count as only one of the six.)

You use this word to specify a file or device whenever you need to perform any operation on either of them. For files that already exist, DQ$ATTACH and DQ$DETACH can add or delete connections. New files are connected with DQ$CREATE.

For example, when your program performs console input and output, the connections for :CI: and :CO: must be on this list. The list permits efficient specification and manipulation of devices or files during execution.

Only objects on this list can be opened or closed, read or written. Use the connection rather than the actual device or file name. During execution, your program may perform these functions on multiple files—but, only six files and devices may be open at one time (not counting :CO:).

Some Series-IV service routines include an "internal" open as part of their operation. When you use such a routine, you may need to close another file or device temporarily to avoid exceeding the limit of six open files-plus-devices at once. However, you may open a physical device more than once because it counts only as one open "file."

Before a file can be read or written (by DQ$READ or DQ$WRITE), it must be connected and opened (by DQ$OPEN). When the activity to that file is completed, it can be closed (by DQ$CLOSE). These four routines can be used only with connections established earlier.

Output devices are created; input devices are attached. For example, workfiles (defined below) and console output :CO: must be created, not attached. Console input is the opposite—:CI: must be attached rather than created.

Buffers

Buffers are areas reserved for expediting disk input/output. A request to buffer a device will be ignored except when :CO: has been redirected to a disk file.

If a series of read operations can be interspersed with calculation, more efficient operation will result. This occurs when the data being read in is not used immediately. Similarly, if a sequence of write operations can be interspersed with calculation, efficiency rises.

When you open a file, buffers are allocated according to your specifications. Your program will read (only), write (only), or update (both). When a file is opened for write, buffer use begins with the first call to DQ$WRITE. When a file is opened for read or update, buffer use begins with the call to DQ$OPEN. When a file is open for update, the most efficient use is clustering the reads (interspersed with calculations) separately from the writes.

You are responsible for indicating the optimal number of buffers for the type of usage you see for a file. For seldom-used files and random I/O, one buffer is best. With sequential console input/output, zero buffers is the correct specification. In all other cases, double buffering is appropriate.

Workfiles

Many programs need temporary files to store immediate results while processing. These files need to be available each time the user logs on. On the Series IV, the designator :WORK: is used to represent these files.

Prior to creating any temporary files, the user must define :WORK: using the LNAME command in the LOGON INT file. This command assigns a directory that allows temporary files to be created and stored under the directory named :WORK:.

Temporary files may be created any number of times. Each time DQ$CREATE is involved with a pointer to the string "6,:WORK:," a new workfile is established.

Workfiles must be created, opened, closed, and detached like any new file. They are automatically deleted when they are detached.

Exception Conditions and Exception Handling

Exceptions, or errors, are detected when an indicated operation cannot be completed. The Series-IV operating system classifies errors as either avoidable or unavoidable. Every system service routine except EXIT returns an error code through a pointer (referred to as excep$p in the descriptions later in this chapter). Your programs can test this code to check if the operation you called for completed successfully. Appendix C contains the standard error names and values.

An error code of zero means your call executed successfully. For example, when you call DQ$OPEN to prepare a file for input/output, you supply a connection number. A zero error code would be returned if the connection representing that file were successfully opened.

A non-zero error code indicates an inappropriate event during a routine's execution. For example, a write operation would fail if the connection representing the desired file indicated that the file had already been closed or detached. A non-zero error code would be returned. Your program should always check for this.

Unavoidable Errors

Unavoidable errors generally arise from environmental conditions outside program control. Examples include insufficient memory for a requested operation, or inability to find an expected file. These errors always return a non-zero value. Often, appropriate program action can be taken. In other instances, nothing can be done until the correct disk is found and inserted, or until the available memory is increased by adjusting other program functions.

Avoidable Errors

Avoidable errors typically are caused by coding errors such as inappropriate parameters or unusable numeric results. Hardware-detected errors, which also fall into this category, include division by zero (interrupt 0), overflow (interrupt 4), and interrupts generated by the 8087 Numeric Data Processor. These errors cause the system's default exception handler to be executed. (See Appendix C.)

However, you may establish your own routine to handle hardware-detected exceptions by using the system routine DQ$TRAP$EXCEPTION and supplying a pointer to your exception-handler. (The state of the stack when your routine gets control is discussed under DQ$TRAP$EXCEPTION.)

One special kind of exception is defined into the system: when a CONTROL-C is typed at the physical console input device by default, the system cancels the current foreground job. You can program your own response to a CONTROL-C via the system routine DQ$TRAP$CC and thereby cause the system to use the CONTROL-C to provide a pointer to your private routine.

Data Types and Register Convention

The descriptions of the system service routines that appear later in this chapter assume data types similar to those of PL/M-86. Your calls to system service routines must supply parameters that meet the following specifications:

BOOLEAN	A BYTE object taking the values TRUE (OFFH) or FALSE (00H). The BOOLEAN specification assumes the following literal definition (in PL/M-86 terms): `DECLARE BOOLEAN LITERALLY 'BYTE' ;`
BYTE	Equivalent to PL/M-86.
CONNECTION	A token representing a connection to a file or device. The CONNECTION specification assumes the following literal definition (in PL/M-86 terms): `DECLARE CONNECTION LITERALLY 'WORD' ;`
DWORD	Four-byte unsigned integer.
POINTER	Equivalent to PL/M-86. Two bytes in the SMALL model of segmentation, four bytes in all others.
SELECTOR	Equivalent to PL/M-86.
STRING	A sequence of bytes the first of which contains the number of bytes following in the sequence, i.e., not including the length byte. A length of zero indicates the null string.
WORD	Equivalent to PL/M-86.

The operating system follows the conventions for interfacing PL/M-86 programs with assembly language programs, saving only registers CS, DS, SS, IP, SP, and BP on a call. Other registers and flags may be used by the operating system routines; upon return to your program, they have no predefined value.

External Procedure Definitions for Series-IV System Service Routines

Introduction

Any module using a service routine must first declare it an external procedure and then link the final object module with the appropriate interface library. Following are the appropriate PL/M-86 declarations with syntax and usage examples for all Series-IV routines.

The routines are presented alphabetically in five categories:

- Exception handling
- File management
- Memory management
- Program control (overlays, exit)
- Utility/command parsing

The file management category has three subclasses:

- Connection to files
- Naming of files
- Use of files

Appendix B lists the routines and parameters in alphabetic order.

Exception Handling Routines

Syntax

```
DQ$DECODE$EXCEPTION:
      PROCEDURE (exception$code, message$p, excep$p) EXTERNAL
      DECLARE exception$code  WORD,
              message$p       POINTER,
              excep$p         POINTER;
      END;
```

Description

exception$code is a word containing an excepion code. *message$p* is a pointer to the 81-byte area (minimum) you declared to receive the error message decoded by the operating system. The first byte of the message is the length of the string. The word whose address is *excep$p* will contain zero unless an unexpected problem in decoding causes a non-zero code to be returned.

The routine returns a string containing the following information:

```
EXCEPTION nnnnH message
```

where

nnnn	is the exception code value.
message	is the exception message.

If the *exception$code* you supply as a parameter is not a recognized system error number, an exception number, not a message, will appear.

Example

```
CALL DQ$CODE$EXCEPTION (ERRNO, @ERRMESS(0), @ERR);
```

Exception Codes Returned

E$OK

Syntax

```
DQ$GET$EXCEPTION$HANDLER:
      PROCEDURE (handler$p, excep$p) EXTERNAL;
      DECLARE handler$p POINTER,
                excep$p      POINTER;
      END;
```

Description

handler$p must point to a four-byte area the system can fill with a long pointer to the current avoidable-exception handler. A four-byte pointer is always returned even if it is called from a program compiled under the SMALL model of segmentation.

If called, this pointer will be the address specified in the last call of DQ$TRAP$EXCEPTION.

Example

```
CALL DQ$GET$EXCEPTION$HANDLER (@WHICH_HANDLER,
@ERR);
```

Exception Codes Returned

EOK, EPTR

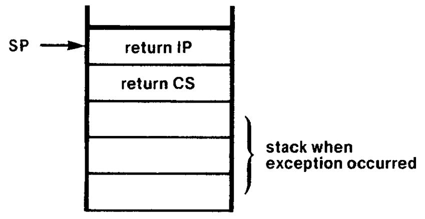
Syntax

```
DQ$TRAP$CC:  PROCEDURE (handler$p, excep$p) EXTERNAL;
             DECLARE handler$p  POINTER,
                     excep$p    POINTER;
             END;
```

Description

Whenver CONTROL-C is pressed at the physical console device, the system executes the default handler or your specially coded routine, whose address is specified as *handler$p* via this system service call.

When your CONTROL-C routine receives control, the registers and flags are the same as in the interrupted program. At that time, the stack looks like the following figure:



121753-8

Write the CONTROL-C routine in assembly language. The program must save the 8086/8088 processor flags and registers and load the DS register with the data segment value of the CONTROL-C routine. Before returning to the interrupted program, the CONTROL-C outline must restore the registers and flags and execute a long return.

The default CONTROL-C handler closes files and terminates program execution. If the key was pressed while the system was performing a system service routine (other than a DQ$READ of :CI:), the routine is completed and the CONTROL-C routine is not executed until just before the system returns to the calling program. If a DQ$READ of :CI: is being serviced, the CONTROL-C handler is executed immediately and the DQ$READ returns an actual count of zero to the calling program. If the job is being processed in BACKGROUND mode, the CONTROL-C request will be ignored.

(If :CI: has been redirected to another device or a disk file, e.g., under SUBMIT, the special meanings of CONTROL-C and CONTROL-D will not be recognized from the SUBMIT file; they will be treated as ordinary characters unless they come from the cold start console.)

Example

```
CALL DQ$TRAP$CC ( SPECIAL_C_PTR,@ERR_C);
```

Exception Codes Returned

E$OK

Syntax

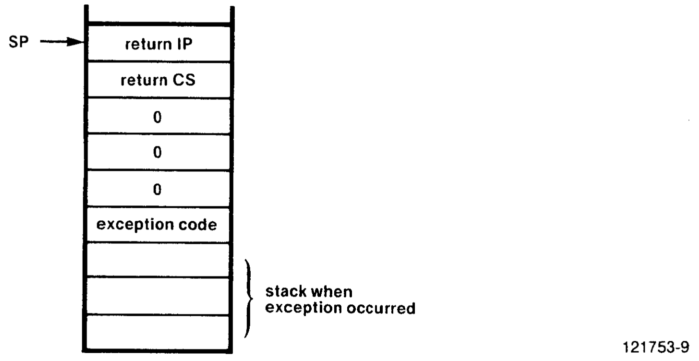
```
DQ$TRAP$EXCEPTION: PROCEDURE (handler$p, excep$p) EXTERNAL;
                   DECLARE handler$p   POINTER,
                           excep$p     POINTER;
                   END;
```

Description

handler$p is the address of a four-byte area containing a long pointer to the entry point of your exception handler. (Programs compiled under the SMALL model of segmentation have no access to CS and thus cannot create the long pointer directly.)

Hardware-detected exceptions will cause the exception handler to be executed. The state of the stack upon entry looks as though the instruction pushed four words and then executed a long call to the exception handler. The first word pushed is the condition code. The next three words are reserved for future use by the operating system and the numeric data processor.

Upon entry to the exception handler, the stack looks like the following diagram:



See Appendix C for exception code values and descriptions.

The default system action displays an error message, closes files, and terminates program execution. Following is the message format:

```
*** EXCEPTION nnnnH error message
    CS:IP = xxxx:yyyy
```

Example

```
EXCEP_ROUT=DQ$TRAP$EXCEPTION (@USER_HANDLER, @ERR);
```

Exception Codes Returned

E$OK

File Management Routines

Connection Routines

Syntax

```
DQ$ATTACH: PROCEDURE (path$p, excep$p) CONNECTION EXTERNAL;
           DECLARE path$p   POINTER,
                   excep$p  POINTER;
           END;
```

Description

path$p points to a string containing a pathname. This string must begin with a number, which denotes the length of the character string. The standard format for defining these strings in the Series-IV operating system is *x*, 'string'. If the pathname consists only of a logical name, the connection created will refer to the directory file indicated by the logical name.

Only input devices and disk files that are not workfiles can be specified via the pathname. Attempting to attach :CO: (or :LP:) will cause an E$SUPPORT exception condition. (However, you may DQ$ATTACH, DQ$CREATE, or DQ$SEEK the Byte Bucket, :BB:.)

The console input device must be attached only via the :CI: pathname. The console output device must be created only via the :CO: pathname.

A maximum of twelve connections can be maintained by Series IV; multiple connections to physical devices (e.g., :LP:) and logical devices (e.g., :BB:, :CI:) are allowed.

DQ$ATTACH operates as a function, i.e., a typed procedure. It returns a connection to an existing file in the variable to the left of the equal sign. If the named file does not exist, the operation will fail and return a non-zero error code at the address pointed to by *excep$*.

Examples

```
DECLARE TAX_CONNECTION WORD, ERR WORD;
TAX_CONNECTION = DQ$ATTACH (@(10, 'FEDTAX.JUN') , @ERR);
```

Exception Codes Returned

E$FNEXIST, E$OK, E$SYNTAX, E$MEM, E$LIMIT, E$TYPE, E$SUPPORT, E$DEVICE$NOT$READY, E$DEVICE$ERROR, E$COMM$ERROR, E$NODE$NOT$READY, E$MARKED$DELETED, E$PARAM

Syntax

```
DQ$CREATE: PROCEDURE (path$p, excep$p) CONNECTION EXTERNAL;
           DECLARE path$p     POINTER,
                   excep$p    POINTER;
           END;
```

Description

DQ$CREATE is a type procedure that returns a connection to a new file. If a file of the same name exists and is not connected, an attempt will be made to delete the existing file and create a new file with this name. This function will fail if the user does not have delete access to the existing file. A non-zero error code will then be returned in the location pointed to by *excep$p*.

path$p points to a string containing a pathname.

Example

```
DECLARE NEW_CONNECTION WORD, ERR WORD;
NEW_CONNECTION = DQ$CREATE ( @(15, '/TAX/NEWTAX.AUG') ,
@ERR) ;
/*A connection number will be created for*/
/*the named file and stored in NEW_CONNECTION.*/
```

Exception Codes Returned

E$SHARE, E$FACCESS, EOK, ESYNTAX, E$SPACE, E$LIMIT, EMEM, ESUPPORT, E$FNEXIST, E$FTYPE, E$PARAM, E$CONNECTION$EXIST, E$DEVICE$NOT$READY, E$COMM$ERROR, E$DEVICE$IO$ERROR, E$NODE$NOT$READY, E$SPACE, E$PARAM, E$FEXIST

Syntax

```
DQ$DELETE: PROCEDURE (path$p, excep$p) EXTERNAL;
           DECLARE path$p     POINTER,
                   excep$p    POINTER;
           END;
```

Description

This routine deletes the file specified by *path$p*. *path$p* points to a string containing a pathname.

File deletion is a logical operation rather than a physical one. The deletion actually occurs when the last connection to the file is detached.

In addition, the supplied pathname must contain a file-name part, and the file to be deleted must not have the write-protect set. A directory file will not be deleted if the directory contains files.

If this operation fails, a non-zero error code will be returned in the location pointed to by *excep$p*.

Example

```
CALL DQ$DELETE ( FILE$PTR, @ERR ) ;
/*The file pointed to by FILE$PTR will be deleted,*/
/*assuming it meets the above conditions.*/
```

Exception Codes Returned

E$FACCESS, E$OK, EPTR, ESUPPORT, E$SYNTAX, E$FNEXIST, E$DEVICE$NOT$READY, E$PARAM, E$DEVICE$IO$READY, E$COMM$ERROR, E$NODE$NOT$READY

Syntax

```
DQ$DETACH: PROCEDURE (conn, excep$p) EXTERNAL;
           DECLARE conn     CONNECTION,
                   excep$p  POINTER;
           END;
```

Description

DQ$DETACH breaks the connection created by DQ$ATTACH or DQ$CREATE. If the connection is open, it is closed before being detached.

conn is a word representing the connection to be detached, i.e., removed from the current list of attached devices or files.

Example

```
CALL DQ$DETACH ( PAY_FILE_CONNECTION, @ERR);
/*The file whose connection is PAY_FILE_CONNECTION*/
/*will be closed and removed from the list of*/
/*connected/attached files, i.e., it will be detached.*/
```

Exception Codes Returned

E$EXIST, E$OK, E$PARAM, E$DEVICE$NOT$READY, E$DEVICE$IO$ERROR, E$COMM$ERROR, E$NODE$NOT$READY E$DETACHED

Syntax

```
DQ$GET$CONNECTION$STATUS:
     PROCEDURE (conn, info$p, excep$p) EXTERNAL;
     DECLARE conn     CONNECTION,
             info$p   POINTER,
             excep$p  POINTER;
     END;
```

Description

This routine supplies the connection status of a connection established earlier. *conn* is a connection established earlier via attach or create. As seen in the following example, the parameter *info$p* points to a structure you have declared to receive the connection data found by this routine.

```
DECLARE INFO STRUCTURE
(OPEN       BOOLEAN,
ACCESS      BYTE,
SEEK        BYTE,
FILE$PTR$   DWORD);
```

The fields just listed have the following interpretations:

OPEN

True if connection is open, otherwise false.

ACCESS

Access privileges of the connection. The rights are granted if the corresponding bit is on.

Data Files		Directory Files	
Bit	**Access**	**Bit**	**Access**
0	delete	0	delete
1	read	1	display
2	write	2	add-entry
3	update		

SEEK

Following are the types of seek supported:

Bit	Seek Types
0	seek forward
1	seek backward
2-7	undefined

FILE$PTR

ırrent position of the file pointer is interpreted as a four-byte unsigned integer (DWORD) representing the number of bytes from the beginning of the file. This file is undefined if the file is not open or if seek backward is not supported.

When the connection you specified is established on a physical device or a logical device, the access value returned depends on the nature of the device. For example, access privilege of the line printer is write. For a disk file with the write protect attributes set, access is read; for workfiles, access is read, write, and update. All other disk files have access privileges of delete, read, and write.

Physical devices and the console do not support any type of seek. For the byte bucket (:BB:), the returned file pointer is 0.

Example

```
CALL DQ$GET$CONNECTION$STATUS (INVENTORY_CONN,
@FILE_STATUS, @ERR);
```

Exception Codes Returned

E$EXIST, E$OK, EPTR, EPARAM, E$COMM$ERROR

Syntax

```
DQ$FILE$INFO:
     PROCEDURE (conn, mode, file$info$p, excep$p) EXTERNAL;
     DECLARE conn,          CONNECTION,
             mode           BYTE,
             file$info$p    POINTER,
             excep$p        POINTER;
     END;
```

Description

DQ$FILE$INFO returns the file information normally associated with user security and accounting.

Following are the input parameters:

> *conn* identifies the connection of a currently attached file.
>
> *mode* indicates whether the file owner is to be identified. Byte Value 0 indicates that the owner name or identification is not to be returned. Byte Value 1 indicates that the owner name or identification is to be returned.
>
> *file$info$p* points to a table used for output.

As seen in the following example, the output of *file$info$p* points to the structure you declare to receive the file information.

```
DECLARE FILE$INFO STRUCTURE
(Owner(15)          STRING,
LENGTH              DWORD,
TYPE                BYTE,
OWNER$ACCESS        BYTE,
WORLD$ACCESS        BYTE,
CREATE$TIME         DWORD,
LAST$MODE$TIME      DWORD,
RESERVED(20)        BYTE);
```

The fields just listed have the following interpretations:

> OWNER is a string that identifies the system name of the owner of the file.
>
> TYPE indicates the type of file. 0 for a data file, 1 for a directory file, and 2-255 reserved.
>
> OWNER$ACCESS and WORLD$ACCESS describe the access rights of the file owner and the world.
>
> > Bit 0 = delete access.
> >
> > Bit 1 = read access (for data files) or display access (for directory files).
> >
> > Bit 2 = write access (for data files) or add_entry access (for directory files).
> >
> > Bit 3 = update (read and write) access.

CREATE$TIME, LAST$MODE$TIME indicates the date and time of creation and last modification for a file. If a file has been created but not modified, the LAST$MODE$TIME should be the same as the CREATE$TIME. A modification consists of a write or update since January 1, 1978. These dates may be decoded into an ASCII string with DQ$DECODE$TIME.

Example

```
CALL DQ$FILE$INFO (conn, mode, file$info$p, excep$p);
```

Exception Codes Returned

EOK, ESUPPORT

Naming Routines

Syntax

```
DQ$CHANGE$ACCESS:
        PROCEDURE (path$p, class, access, excep$p) EXTERNAL;
        DECLARE path$p      POINTER,
                class       BYTE,
                access      BYTE,
                excep$p     POINTER;
        END;
```

Description

DQ$CHANGE$ACCESS changes the owner and world access rights to a file. The privilege to use this primitive is assured for the owner of the file. The granting of this privilege to other users is operating-system dependent. If the privilege is not granted, the error E$FACCESS is supported. E$FNEXIST indicates the file does not exist; E$SUPPORT indicates an attempt to change the access rights of a non-disk file. The access rights of the file will be changed immediately but will not affect connections to the file until they are detached.

Following are the input parameters:

path$p points to a string containing the pathname of the file whose access rights are to be changed.

class specifies the class of users whose access rights are to be changed. 0 indicates the owner, 1 the world, and 2-255 reserved.

access specifies the type of access to be granted to the class of file users specified. If all bits are set to 0, the specified user's access to the file will be denied. If any bits are set to 1, the following access is granted:

Bit 0 = delete access (for data files and directory files).

Bit 1 = read access (for data files), display (for directory files).

Bit 2 = write access (for data files), add_entry access (for directory files).

Bit 3 = update access (read and write).

No output is returned by this call.

Example

```
CALL DQ$CHANGE$ACCESS (@FILE_NAME, CLASS, ACCESS, @ERR);
```

Exception Codes Returned

EOK, EMEM

Syntax

```
DQ$CHANGE$EXTENSION:
      PROCEDURE (path$p,  extension$p,  excep$p) EXTERNAL;
      DECLARE path$p       POINTER,
              extension$p  POINTER,
              excep$p      POINTER;
      END;
```

Description

DQ$CHANGE$EXTENSION replaces any existing extension on the file name with the supplied extension.

path$p points to a string containing the pathname to be changed. *extension$p* points to a three-character extension that is to become the extension in the pathname. These characters may not be delimiters. This procedure changes only the specified string and performs no file operations whatsoever.

If the first character addressed by *extension$p* is a blank, any prior extension of the file-name, including the trailing period, will be deleted. (Trailing blanks are allowed, i.e., the third character or both the second and third characters of the new extension may be blanks.)

Examples

1.
```
CALL DQ$CHANGE$EXTENSION (@(8, 'TASK.QRY'),
@('ANS'), @ERR$P);
/*Filename string will be changed from*/
/*TASK.QRY to TASK.ANS*/
```

2.
```
CALL DQ$CHANGE$EXTENSION ( FILE$PTR, @('OBJ'), EXCEP$P);
/*This will change the extension on the filename*/
/*pointed to by FILE$PTR to be .OBJ*/
```

Exception Codes Returned

EOK, ESTRING$BUF, E$PTR, E$SYNTAX

Syntax

```
DQ$RENAME: PROCEDURE (old, new$p, excep$p) EXTERNAL;
           DECLARE old$p    POINTER,
                   new$p    POINTER,
                   excep$p  POINTER;
           END;
```

Description

DQ$RENAME changes the name of a file within its parent directory. The new file name is not restricted to a single path component; however, the old and new pathnames should differ only in the last component.

old$p and *new$p* are pointers to the strings containing the existing pathname and the new pathname, respectively.

An exception condition occurs if a file with a new name already exists or if the file to be renamed has the write-protect attribute set. Renaming a file on which a connection is established is valid, and the connection does not need to be re-established.

Example

```
CALL DQ$RENAME (@FILE_PTR(3), @(9, 'TERMS.NOV'), @ERR);
```

Exception Codes Returned

E$FACCESS, E$FEXIST, E$FNEXIST, E$OK, EPTR, ESUPPORT, E$SYNTAX, E$MEM, E$FTYPE, E$DEVICE$NOT$READY, E$COMM$ERROR, E$NODE$NOT$READY, E$PARAM

Usage Routines

Syntax

```
DQ$CLOSE: PROCEDURE (conn, excep$p) EXTERNAL;
          DECLARE conn     CONNECTION,
                  excep$p  POINTER;
          END;
```

Description

DQ$CLOSE waits for completion of input/output operations (if any) taking place on the file, ensures output buffers are empty, and frees buffers. Once closed, a connection may be either re-opened or detached. CLOSE does not truncate the file; the original extent (or the new extent enlarged by writes) is maintained.

conn represents a connection established earlier using attach or create and opened via OPEN.

Programs that attach :CI: and create :CO: should open, close and detach them. :CI: and :CO: do not count towards the limit of open files.

Example

```
CALL DQ$CLOSE (TAX;_CONNECTION, @ERR);
/*The file whose connection number is in*/
/*TAX_CONNECTION will be closed.*/
```

Exception Codes Returned

E$EXIST, E$NOPEN, E$PARAM, E$OK,
E$DEVICE$NOT$READY, E$DEVICE$IO$ERROR,
E$COMM$ERROR, E$MODE$NOT$READY, E$SPACE

Syntax

```
DQ$OPEN: PROCEDURE (conn, access, num$buf, excep$p) EXTERNAL;
         DECLARE conn        CONNECTION,
                 access      BYTE,
                 num$buf     BYTE,
                 excep$p     POINTER,
         END;
```

Description

conn represents a connection established earlier via attach or create.

Value	Type
1	read access only
2	write access only
3	update (both read and write)

num$buf, which indicates the optimal number of buffers, should be 0, 1, or 2. Zero means that no buffering should occur; physical I/O should occur during a DQ$READ or DQ$WRITE. For seldom-used files, *num$buf* should be 1. In all other cases it should be 2 for double buffering. If program computation cannot be interspersed as described earlier in this chapter, *num$buf* should be 0 to maximize performance.

Files can be opened "externally" using this routine or "internally" as part of the operation of other system service routines. The limit of six open files and devices includes those opened internally by the system. DQ$ATTACH, DQ$CREATE, DQ$TRUNCATE, and entry to DEBUG-88 all perform an "internal open." Entering DEBUG-88 internally opens :CI:.

Since it is possible to establish multiple connections on the same device, multiple opens of such a device are also permitted. The device still counts as only one open device on the list of six. The console input device and output devices do not count toward this limit.

If *access* is write or update, the file represented by the connection must not have the write-protect or format attributes set. The file pointer is set to 0, i.e., the beginning of the file. If the next access to this file is write, writing begins at the first byte, destroying earlier contents. You must read or seek to the end of the file first, and then write to it to append information.

The use of DQ$OPEN must not violate physical limitations; e.g., the line printer must not be opened for read or update.

Example

```
CALL DQ$OPEN (EMPLOYEE_CONN, 3, 2, @ERR);
```

Exception Codes Returned

E$EXIST, E$FACCESS, EOK, EPARAM, E$OPEN, E$SIX,
E$SHARE, E$FNEXIST

Syntax

```
DQ$READ: PROCEDURE (conn, buf$p, count, except$p) WORD EXTERNAL;
         DECLARE conn      CONNECTION,
                 buf$p     POINTER,
                 count     WORD,
                 excep$p   POINTER;
         END;
```

Description

conn represents an open connection established earlier via attach or create. *buf$p* points to a buffer area at least count bytes long which you have declared to receive the data read.

This routine is used as a function, i.e., a typed procedure returning the number of bytes actually transferred. This number will equal *count* unless an error occurred or an end-of-file was encountered.

count bytes are read from the current location of the file pointer and placed in your buffer. If the procedure returns a value less than *count* and an exception code of E$OK, end-of-file was encountered.

If your buffer is not long enough to receive the number of bytes requested, this routine will over-write the memory locations that follow the buffer.

DQ$READ will not recognize CONTROL-C and CONTROL-D as having any special meaning if the console has been assigned to a disk or device other than the cold start console. (See also DQ$TRAP$CC.)

Example

```
DECLARE ACTUAL WORD ENTRIES (256) BYTE, ERR WORD;
ACTUAL = DQ$READ (JOURNAL_CONN, @ENTRIES(0), 256,
@ERR);
```

Exception Codes Returned

E$EXIST, E$NOPEN, EOK, EOWRITE, E$PARAM, E$PTR

Syntax

```
DQ$SEEK: PROCEDURE (conn,mode, offset, excep$p) EXTERNAL;
         DECLARE conn      CONNECTION,
                 mode      BYTE,
                 offset    DWORD,
                 excep$p   POINTER;
         END;
```

Description

conn represents a currently open connection established earlier via attach or create.

mode indicates the type of seek required.

Value	Type
1	move file pointer back by offset
2	set pointer to offset
3	move file pointer forward by offset
4	move file pointer to end-of-file minus offset

offset forms a four-byte unsigned integer (DWORD) representing the number of bytes required to move the file pointer.

If the seek goes beyond the end-of-file and a subsequent write occurs, a file opened for write or update will be extended with nulls. For a file opened for read, such a seek will position the pointer beyond the end-of-file. If a seek would move the pointer to a position before the start of the file, the pointer is set to the beginning of the file. Seeks are invalid on connections to physical devices or the console.

Example

```
CALL DQ$SEEK (IONS_CONN, 3, 22, @ERR);
```

This call does a seek from the current position forward—by an offset of 22 $\times$ 2^{16} bytes—on the file whose connection is IONS_CONN.

Exception Codes Returned

E$EXIST, E$NOPEN, EOK, EPARAM, E$SUPPORT, E$SPACE, E$DEVICE$IO$ERROR, E$DEVICE$NOT$READY, E$COMM$ERROR, E$NODE$NOT$READY

Syntax

```
DQ$SPECIAL: PROCEDURE (type, parameter$p, excep$p) EXTERNAL;
            DECLARE type           BYTE,
                    parameter$p    POINTER,
                    excep$p        POINTER;
            END;
```

Description

This routine determines whether subsequent console input is transparent or line-edited.

type = 1 indicates the subsequent console input is transparent, i.e., not line-edited.

type = 2 indicates the subsequent console input will be line-edited. The initial default, when a job begins, is type two.

type = 3 is identical to a type of 1 except that a DQ$READ of :CI: will return only the single character already in the system buffer.

type = 4 sets the word pointed to by PARAM_P to 1. This indicates to the calling program that it is running on a Series-IV.

parameter$p must point to a connection representing a DQ$ATTACH of :CI:.

:CI: can be assigned to a disk file with a SUBMIT system call. This routine returns E$SUPPORT if type 1 or type 3 is specified; this occurs even if :CI: is temporarily restored to the cold start console via control E.

A call to this routine changes :CI: from line-edited to transparent and causes all characters currently in the physical console buffer to be discarded. The key buffer can be cleared by reading 32 characters in polling (DQ$SPECIAL type 3) made from :CI: and discarding the results.

Example

```
CALL DQ$SPECIAL (1, @CI_CONN, @ERR);
```

Exception Codes Returned

EOK, EEXIST, E$PARAM, E$PTR, E$SUPPORT

Syntax

```
DQ$TRUNCATE: PROCEDURE (conn, excep$p) EXTERNAL;
             DECLARE conn     WORD,
                     excep$p  POINTER;
             END;
```

Description

This routine truncates the file represented by *conn* at the current file pointer and frees all previously allocated disk space beyond that pointer value. (If the pointer is at the end-of-line or past the end-of-file, truncation has no effect.)

conn represents a connection established earlier via attach or create and currently open for write or update.

Example

```
CALL DQ$TRUNCATE (INTERIM_CONN, @ERR);
```

Exception Codes Returned

EOK, EEXIST, E$NOPEN, E$SHARE, E$SPACE,
E$PARAM, E$MEM, E$OPEN$MODE, E$SUPPORT,
E$DEVICE$NOT$READY, E$NODE$NOT$READY

Syntax

```
DQ$WRITE: PROCEDURE (conn, buf$p, count,excep$p) EXTERNAL;
          DECLARE conn      CONNECTION,
                  buf$p     POINTER,
                  count     WORD,
                  excep$p   POINTER;
          END;
```

Description

conn represents an open connection established earlier via attach or create. Access must be write or update.

buf$p points to the start of the data to be written out.

count is the number of bytes to be written.

If the *count* exceeds the remaining length of your buffer, the contents of memory locations following the buffer will be written to the device or file represented by the connection you supplied.

Writing begins at the current value of the file-pointer, which is 0 if no prior reads, seeks, or writes to this file have occurred.

A write to a pre-existing file you have opened will destroy earlier contents unless the file-pointer is first positioned (by a seek) at the end-of-file or after the end-of-file. A subsequent close, however, does not truncate the file; the original extent (or the new extent enlarged by writes) is maintained.

Example

```
ALL DQ$WRITE (INVENTORY_CONN,@PHYSICAL, 256, @ERR);
```

Exception Codes Returned

E$EXIST, E$NOPEN, EOK, EPARAM, EMEM, EOPEN$MODE, E$SPACE, E$DEVICE$NOT$READY, E$DEVICE$IO$READY, E$COMM$ERROR, E$NODE$NOT$READY, E$DEVICE$IO$ERROR

NOTE

A DQ$WRITE can return the error message E$OK when the DQ$READ call actually results in I/O errors. To detect this error, a subsequent DQ$READ, DQ$WRITE, DQ$SEEK, DQ$CLOSE, or DQ$TRUNCATE call should be made to the same file connection.

Memory Management Routines

Syntax

```
DQ$ALLOCATE: PROCEDURE (size, excep$p) SELECTOR EXTERNAL;
             DECLARE size     WORD,
                     excep$p POINTER;
             END;
```

Description

size is the number of bytes of memory desired. A size of zero means a request for 64K bytes. If enough memory is available, this function returns a SELECTOR representing the base of the acquired memory block—that is, the segment part of the pointer to the acquired area. (The offset of this pointer is zero.) If the operation fails, this SELECTOR will be 0FFFFH.

When a program allocates memory using DQ$ALLOCATE, the program must not try to access more memory than that allocated by the DQ$ALLOCATE call.

DQ$ALLOCATE is used as a function, i.e., to the right of an equal sign in a PL/M-86 assignment statement. The variable to the left of the equal sign is filled with the segment token.

Example

```
PAY_REC_STRUC_BASE = DQ$ALLOCATE (12, @ERROR_PAY);
IF (PAY_REC_STRUC_BASE = 0FFFFH) THEN
    CALL MY_ERRCHK (PAY_REC_STRUC_BASE, LESS_MEM);
IF (ERROR_PAY 0) THEN
    CALL MY_ERRCHK (ERROR_PAY, MEM_ERR);
.....
.....
```

Exception Codes Returned

EOK, EMEM

Syntax

```
DQ$FREE: PROCEDURE (segment, excep$p) EXTERNAL;
         DECLARE segment  SELECTOR,
                 excep$p  POINTER;
         END;
```

Description

segment is a SELECTOR representing a memory segment acquired earlier from DQ$ALLOCATE. The indicated segment is freed. This liberation should be done at the end of the task that allocated the segment, or whenever the segment will no longer be needed.

DQ$EXIT automatically frees all memory segments allocated by DQ$ALLOCATE. Therefore, you need only DQ$FREE a segment when the program needs to reclaim the space for another purpose.

Once a program has freed a particular segment, do not access memory in that freed segment.

Example

```
CALL DQ$FREE (PAY_REC_STRUC_BASE, @ERROR_LESS)
```

Exception Codes Returned

EOK, EBAD$SEGMENT

Syntax

```
DQ$GET$SIZE: PROCEDURE (segbase, excep$p) WORD EXTERNAL;
             DECLARE segbase SELECTOR,
                     excep$p POINTER;
             END;
```

Description

This function returns the size of the segment in bytes; zero means 64K bytes. This segment must have been previously allocated with the DQ$ALLOCATE routine.

segbase is a SELECTOR for a memory segment.

The loader uses DQ$ALLOCATE to get the memory it requires to load your program from the memory manager. When you link your program, you can specify an "expanding segment," which means the size will be determined when the program is loaded. The size depends upon the amount of memory available.

Relocatable PL/M-86 programs compiled under the SMALL model of segmentation can use an expanding data segment whose size is determined with a statement that has the following form:

```
SIZE = DQ$GET$SIZE (STACKBASE, @EXCEP);
```

In SMALL model programs, the stack segment base and the data segment base are the same. Thus, the above PL/M-86 statement passes the token representing the data segment base and obtains the data segment size.

Example

```
DECLARE ARRAY_SIZE WORD;
ARRAY_SIZE = DQ$GET$SIZE (ARRAY_BASE, @ERR);
```

Exception Codes Returned

EOK, EBAD$SEGMENT

Syntax

```
DQ$RESERVE$IO$MEMORY:
   PROCEDURE (number$files, number$buffers, excep$p) EXTERNAL;
   DECLARE number$files      WORD,
           number$buffers    WORD,
           excep$p           POINTER;
      END;
```

Description

DQ$RESERVE$IO$MEMORY informs the operating system of the maximum number of files to be attached, and the maximum number of buffers to be requested during the execution of a particular program. The call requests the system to reserve enough memory to assure that the Creates, Attaches and Opens will be successful. The default value for the two variables specified in this function is zero. Furthermore, you may assume that at least twelve Attaches and six Opens will be supported by calling DQ$RESERVE$IO$MEMORY—although some operating systems may allow for more.

You can use this to anticipate calls to DQ$ATTACH and DQ$OPEN which you will need to make. By warning the operating system of these calls, the system can reserve memory so that intervening calls to DQ$ALLOCATE do not prevent the Attaches and Opens from being successful. Successive calls to this function are legal; they simply change the current number of buffers requested for the corresponding program. A request to increase the number of buffers can fail due to a lack of memory, especially if calls to DQ$ALLOCATE have been made since the previous call to DQ$RESERVE$IO$MEMORY. The call to DQ$RESERVE$IO$MEMORY should occur before the first call to DQ$ALLOCATE, thereby maximizing the probability that it will be successful.

The input parameters are *number$files* and *number$buffers. number$files* is the maximum number of files to be attached at any one time. If this value is exceeded, the application takes the responsibility for ensuring that the additional memory is preconfigured or available for allocation when the calls to DQ$ATTACH and DQ$CREATE are made. *number$buffers* is the maximum number of buffers to be required for any concurrent set of open files. *number$buffers* limits the total number of *num$buffers* parameters allowable in any set of calls to DQ$OPEN for which corresponding calls to DQ$CLOSE have not been made. If this parameter is exceeded, the application takes the responsibility of ensuring that the additional memory is available for allocation when the calls to DQ$OPEN are made.

For compatibility when porting to other operating systems, DQ$RESERVE$IO$MEMORY allows the maximum number of connections and opens allowed by that system (for example RMX-86) to be used. The default case in the Series IV is a maximum of 12 connections and 6 opens.

Example

```
CALL DQ$RESERVE$IO$MEMORY (NUMBER_FILES, NUMBER_BUFFERS,
@ERR);
```

Exception Codes Returned

EOK, EMEM

Program Control Routines

Syntax

```
DQ$EXIT: PROCEDURE (completion$code) EXTERNAL;
         DECLARE completion$code  WORD;
         END;
```

Description

Exit terminates a job. All files are closed and all resources are freed. If ISIS-IV was in control, it prompts for another command.

The *completion$code* indicates whether termination was normal. Following are its values and interpretations:

0 — normal termination
1 — warning messages were issued
2 — errors were detected
3 — fatal errors were detected
4 — job aborted

completion code has no exception pointer argument because it never generates an exception.

Example

```
CALL DQ$EXIT (COMPL);
```

Exception Codes Returned

none

Syntax

```
DQ$OVERLAY: PROCEDURE (name$p, excep$p) EXTERNAL;
            DECLARE name$p     POINTER,
                    excep$p    POINTER;
             END;
```

Description

This routine causes the loading of the overlay whose name is the string pointed to by *name$p*. Only one level of overlays is allowed. This routine may be called only from the root (non-overlaid) phase.

You must define the overlay name with the LINK86 OVERLAY control. The name must not exceed 40 characters. The string used in the call must match the name used in LINK86.

(See the *iAPX 86, 88 Family Utilities User's Guide*, 121616, for a full discussion of overlays.)

Example

```
CALL DQ$OVERLAY (@(10 'MYPROG.OV2'), @ERR);
```

Exception Codes Returned

EOK, EEXIST, E$PARAM, E$SIX, E$SYNTAX, E$UNSAT, E$ADDRESS, E$BAD$FILE

Utility and Command Parsing Service Routines

Syntax

```
DQ$DECODE$TIME: PROCEDURE (dt$p, excep$p EXTERNAL;
                DECLARE dt$p       POINTER,
                        excep$p    POINTER;
                END;
```

Description

DQ$DECODE$TIME decodes the operating system dependent time and date DWORD into ASCII data and time strings. It returns the current date and time in binary DWORD format and/or as a decoded ASCII string.

dt$p is a pointer to a user declared structure of the following form:

```
DECLARE DT STRUCTURE
 (SYSTEM$TIME DWORD,
   DATE(8)   BYTE,
   TIME(8)   BYTE);
```

system$time, the output parameter, is an operating-system-dependent formatted DWORD containing the time and date. If *system$time* is zero, the system clock is first read to obtain the current date and time. If *system$time* is non-zero, it is simply decoded into the ASCII date and time string.

system$time will contain the binary format of the current date and time as selected by the input value zero. The specified format is in seconds, beginning with January 1, 1978.

DATE has the form MM/DD/YY for month, day, and year. TIME has the form HH:MM:SS for hours, minutes, and seconds. The value for hours is in the range 0-23.

Example

```
CALL DQ$DECODE$TIME (dt$p, excep$p);
```

Exception Codes Returned

EOK, ESUPPORT

Syntax

```
DQ$GET$ARGUMENT: PROCEDURE (argument$p, excep$p) BYTE EXTERNAL;
                 DECLARE argument$p POINTER,
                         excep$p    POINTER;
                 END;
```

Description

This function returns the arguments in the command line or user-supplied buffer. Each successive call returns the next argument.

argument$p points to an 81-byte area you have declared to receive a strong argument from the command tail (see the section on command tail arguments).

This typed procedure is used as a function (i.e., on the right side of an equal sign in a PL/M-86 assignment statement). The variable to the left of the equal sign is filled with the delimiter found by DQ$GET$ARGUMENT, as shown in the examples below. If the exception code returned is zero, the call functioned properly. The possible delimiters include:

,) (= # ! $ % ’ \
− + − & |] [> < : or DEL (RUB OUT)

or have a hexadecimal value of 0-20H for ASCII characters. For non-ASCII characters, possible delimiters also include hexadecimal values $\geqslant$ 80H.

The command line is pre-scanned when your program is invoked. At that time, the system makes the following changes to the command line before it is saved in a system buffer:

1. Each continuation line sequence is converted to a blank. A continuation line sequence begins with an "&" and ends with the line terminator.
2. A comment is removed entirely. A comment begins with a ";" and ends with the character preceding the line terminator.
3. Any DEBUG commands preceding the pathname are removed.

The following rules apply to the arguments and delimiters returned by DQ$GET$ARGUMENT:

1. Lowercase alphabetic characters, except inside quotes, are converted to uppercase.
2. Multiple adjacent blanks separating two arguments are treated as one blank. One or more blanks adjacent to any other delimiter are ignored. A tab is treated like a blank and returned as a blank.
3. Strings enclosed within a pair of matching quotes are considered literals and are not scanned for interpretation. The enclosing quotes are not returned as part of the argument. Quotes may be used in a quoted string by doubling the quote.

The DQ$SWITCH$BUFFER routine may be used to get arguments from a user-supplied buffer. The command line pre-scan is not performed on this buffer.

Examples

The following examples illustrate the argument returned by calls to DQ$GET$ARGUMENT:

A)
```
PLM86.86 LINKER.PLM PRINT (:LP:) NOLIST
```

Argument	Delimiter
8,‘PLM86.86’	. .
10,’LINKER.PLM’	. .
5,‘PRINT’	‘(’
4,‘:LP:’	‘)’
6,‘NOLIST’	<cr>

B)
```
PLM86.86 MODULE.SRC PRINT(/VOL1/THISIS.IT)OPTIMIZE(0)
TITLE ('MY MODULE')
```

Argument	Delimiter
8,‘PLM86.86’	‘ ’
10,‘MODULE.SRC’	‘ ’
5,‘PRINT’	‘(’
13,‘/VOL1/THISIS.IT’	‘)’
8,‘OPTIMIZE’	‘(’
1,‘0’	‘)’
5,TITLE	‘(’
9,MY MODULE	‘)’
0	<cr>

C)
```
LINK86.86/VOL4/X.OBJ, LLIN(MODL), SYSTEM,LIB, PUBLICS &
(/VOL3/FUNNY.LIB(MOD1)) MAP; my link command
```

Argument	Delimiter
9,‘LINK.86’	‘ ’
9,‘/VOL4/X.OBJ’	‘.’
4,‘LLIB’	‘(’
0	‘)’
10,‘SYSTEM.LIB’	‘.’
7,‘PUBLICS’	‘(’
13,‘/VOL3/FUNNY.LIB’	‘(’
4,‘MOD1’	‘)’
0	‘)’
3,‘MAP’	<cr>

Example

```
DECLARE DELIM_SCAN BYTE, ERR WORD;
DECLARE NEXT_ARG STRUCTURE
        (LENGTH BYTE, ARG (80) BYTE);
DELIM_SCAN=DQ$GET$ARGUMENT
        (@NEXT_ARG.LENGTH, @ERR);
```

Exception Codes Returned

EOK, ESTRING$BUF, E$PTR

Syntax

```
DQ$GET$SYSTEMS$ID: PROCEDURE (id$p, excep$p) EXTERNAL;
                   DECLARE id$p     POINTER,
                           excep$p  POINTER;
                   END;
```

Description

This routine returns a string identifying the operating system. *id$p* must point to a 21-byte buffer you define in your program.

Example

```
CALL DQ$GET$SYSTEM$ID (@ID, @ERR);
```

Exception Codes Returned

EOK, EPTR

Syntax

```
DQ$GET$TIME: PROCEDURE (dt$p, excep$p) EXTERNAL;
             DECLARE dt$p     POINTER,
                     excep$p  POINTER;
             END;
```

Description

This routine will return the system date, which can be set by the DATE command.

dt$p must be a pointer to a user structure of the following form:

```
DECLARE DT STRUCTURE
(DATE(8) BYTE, TIME(8) BYTE);
```

DATE has the form MM/DD/YY for month, day, and year. TIME has the form HH/MM/SS for hours, minutes, and seconds.

Example

```
CALL DQ$GET$TIME (@PT, @ERR)
```

Exception Codes Returned

E$OK

Syntax

```
DQ$SWITCH$BUFFER: PROCEDURE (buffer$p, excep$p) WORD EXTERNAL;
                  DECLARE buffer$p POINTER,
                          excep$p POINTER;
                  END;
```

Description

This routine is used as a function.

The offset value (*buffer$p*) returned from the invocation of DQ$GET$ARGUMENT points to the buffer position of the previous buffer. By keeping track of all previous buffer position pointers you can switch between user-supplied buffers when parsing a command line.

When this routine is first invoked, you cannot switch back to arguments from the input command buffer.

Example

```
DECLARE NEXT_COUNT WORD, ERR WORD;
NEXT_COUNT=DQ$SWITCH$BUFFER (@ARGLIST, @ERR);
```

Exception Codes Returned

E$OK

APPENDIX A
CLI COMMAND SYNTAX

```
ACCESS  [filename [SET {USER SPEC  DATA ACCESS SPEC} ] ]
                                   DIR ACCESS SPEC

                           [M  [{{B | S | ON} [MDY [HMS]]}]                   ]
                           [   [{          T             }]                   ]
ARCHIVE old-dir TO new-dir [C   {{B | S | ON} [MDY [HMS]]}                    ] [AND ...] {U}
                           [    {          T             }                    ]           {Q}
                           [OW        owner name                              ]
                           [F         pathname                                ]

BACKGROUND  pathname [(a-parameters)]  ...  [{  LOG}]
                                            [{NOLOG}]

BATCH  pathname

CANCEL {BACKGROUND                                   }
       {REMOTE  queue {(jobname)  (#jobnumber)}       }

CHOWNER  filename  TO  username

CHPASS  username

COPY  sourcefilename  TO  destinationfilename  [{UPDATE}]
                                               [{QUERY }]

COUNT  n

      [ commands ]

      [{WHILE} argument { = } argument ] ...
      [{UNTIL}          {<>}           ]
      [ commands                       ]

      END

CREATEDIR  pathname

DELETE [pathname {DIR                  }]
       [         {QUERY                }]
       [         {:SP:  / request name }]

DIR [{pathname}[FOR filename] [EXPANDED] [TO pathname]]
    [{   /    }                                        ]

DISMOUNT  device name

EXIT  [(argument)]

EXPORT  pathname [parameters]  TO  queue [{  LOG}]
                                         [{NOLOG}]

FILL {ON }
     {OFF}
```

```
FORMAT physical-device volume-name [{FNODES (number) NOINIT RESERVE
        (reserve-option, ...)}] ...

          ⎧UP    iNDX-source-pathname TO destination-pathname   ⎫ ⎧UPDATE⎫
⎧S4FPRT⎫ ⎨EXIT                                                 ⎬ ⎨EXIT  ⎬
⎩S2FPRT⎭ ⎩DOWN  [disk-dir] ISIS-source-pathname                 ⎭ ⎩QUERY ⎭
                TO iNDX-destination-pathname

ICOPY ⎧READ ISIS-source-pathname TO iNDX-destination-pathname⎫ ⎧QUERY ⎫
      ⎩WRITE iNDX-source-pathname TO destination-pathname    ⎭ ⎩UPDATE⎭

IF argument ⎧= ⎫ argument
            ⎩<>⎭

   commands
        [ORIF   argument ⎧= ⎫ argument ]
                commands ⎩<>⎭

        [ELSE   commands]

   END

IMPORT FROM queue [,queue] ... [TO BACKGROUND]

LNAME ⎧DEFINE logical name FOR pathname [UPDATE]⎫
      ⎨REMOVE logical name                      ⎬
      ⎩PATH                                     ⎭

LOG ⎧:BB:    ⎫
    ⎩pathname⎭

LOGOFF

LOGON username [⎧INIT (filename)⎫]
               [⎩NONIT          ⎭]

MOUNT device-name

OPEN pathname

PDSCOPY ⎧READ (disk-directory) PDS-source TO iNDX-destination⎫ ⎧QUERY ⎫
        ⎩WRITE iNDX-source TO PDS-destination                ⎭ ⎩UPDATE⎭

QUEUE

READ variable-name   [, ...]

REGION

RENAME old-pathname TO file-name [UPDATE]
```

```
REPEAT

    [ commands ]

     [ { WHILE } ]  argument  { =  }  argument
     [ { UNTIL } ]            { <> }

     commands

                                                   { FORMAT }
SDCOPY source-device-name[TO dest-device-name]     { REPEAT }  ...[COMPARE]
                                                   { VERIFY }     [REPEAT]

SET variable-name TO ["] value ["]

SPACE /volume-name

SUBMIT pathname [(a-parameters)] ... [ {   LOG } ]
                                     [ { NOLOG } ]

SYSTAT [ { QUEUE } [(queuename [,...])] ] [TO pathname] [EXPAND] [ALL]
       [ { MYJOB }                      ]

TIME

USERDEF { DEFINE username [ID userid] [DIR filename] }
        { REMOVE username                            }

USERS /volume-name

VERIFY device-name [FIX]

VIEW pathname
```

APPENDIX B PARAMETERS AND SYSTEM SERVICE ROUTINES

Table B-1 lists the system service routines in alphabetical order for study or reference. Table B-2 alphabetically lists the parameters used by the service routines.

Table B-1. Alphabetical List of Series-IV Service Routines

```
ALLOCATE
ATTACH
CHANGE$ACCESS
CHANGE$EXTENSION
CLOSE
CREATE
DECODE$EXCEPTION
DECODE$TIME
DELETE
DETACH
EXIT
FILE$INFO
FREE
GET$ARGUMENT
GET$CONNECTION$STATUS
GET$EXCEPTION$HANDLER
GET$SIZE
GET$SYSTEM$ID
GET$TIME
OPEN
OVERLAY
READ
RENAME
RESERVE$IO$MEMORY
SEEK
SPECIAL
SWITCH$BUFFER
TRAP$CC
TRAP$EXCEPTION
TRUNCATE
WRITE
```

```
DQ$ALLOCATE:     PROCEDURE (size, excep$p) TOKEN EXTERNAL;
       DECLARE size            WORD,
               excep$p         POINTER;
       END;
```

```
DQ$ATTACH:    PROCEDURE (path$p, excep$p) CONNECTION EXTERNAL;
       DECLARE size            POINTER,
               excep$p         POINTER;
       END;
```

```
DQ$CHANGE$ACCESS:     PROCEDURE (path$p, class, access, excep$p) EXTERNAL;
       DECLARE path$p          POINTER,
               class           BYTE,
               access          BYTE,
               excep$p         POINTER;
       END;
```

```
DQ$CHANGE$EXTENSION:     PROCEDURE (path$p, extension$p, excep$p) EXTERNAL;
       DECLARE path$p          POINTER,
               extension$p     POINTER,
               excep$p         POINTER;
       END;
```

Table B-1. Alphabetical List of Series-IV Service Routines (Cont'd.)

```
DQ$CLOSE:   PROCEDURE (conn, excep$p) EXTERNAL;
        DECLARE conn            CONNECTION,
                excep$p         POINTER;
        END;
```

```
DQ$CREATE:   PROCEDURE (path$p, excep$p) CONNECTION EXTERNAL;
        DECLARE path$p          POINTER,
                excep$p         POINTER;
        END;
```

```
DQ$DECODE$EXCEPTION:   PROCEDURE (exception$code, message$p, excep$p)
                       EXTERNAL;
        DECLARE exception$code  WORD,
                message$p       POINTER,
                excep$p         POINTER;
        END;
```

```
DQ$DECODE$TIME:   PROCEDURE (dt$p, excep$p) EXTERNAL;
        DECLARE dt$p            POINTER,
                excep$p         POINTER;
        END;
```

```
DQ$DELETE:   PROCEDURE (path$p, excep$p) EXTERNAL;
        DECLARE path$p          POINTER,
                excep$p         POINTER;
        END;
```

```
DQ$DETACH:   PROCEDURE (conn, excep$p) EXTERNAL;
        DECLARE conn            CONNECTION,
                excep$p         POINTER;
        END;
```

```
DQ$EXIT:   PROCEDURE (completion$code) EXTERNAL;
        DECLARE completion$code WORD;
        END;
```

```
DQ$FILE$INFO:   PROCEDURE (conn, mode, file$info$p, excep$p) EXTERNAL;
        DECLARE conn            CONNECTION,
                mode            BYTE,
                file$info$p     POINTER,
                excep$p         POINTER;
        END;
```

```
DQ$FREE:   PROCEDURE (segment, excep$p) EXTERNAL;
        DECLARE segment         TOKEN,
                excep$p         POINTER;
        END;
```

```
DQ$GET$ARGUMENT:   PROCEDURE (argument$p, excep$p) BYTE EXTERNAL;
        DECLARE argument$p      POINTER,
                excep$p         POINTER;
        END;
```

```
DQ$GET$CONNECTION$STATUS:   PROCEDURE (conn, info$p, excep$p) EXTERNAL;
        DECLARE conn            CONNECTION,
                info$p          POINTER,
                excep$p         POINTER;
        END;
```

```
DQ$GET$EXCEPTION$HANDLER:   PROCEDURE (handler$p, excep$p) EXTERNAL;
        DECLARE handler$p       POINTER,
                excep$p         POINTER;
        END;
```

Table B-1. Alphabetical List of Series-IV Service Routines (Cont'd.)

```
DQ$GET$SIZE:    PROCEDURE (segbase, excep$p) WORD EXTERNAL;
        DECLARE segbase         TOKEN,
                excep$p         POINTER;
        END;
```

```
DQ$GET$SYSTEM$ID:    PROCEDURE (id$p, excep$p) EXTERNAL;
        DECLARE id$p            POINTER,
                excep$p         POINTER;
        END;
```

```
DQ$GET$TIME:    PROCEDURE (dt$p, excep$p) EXTERNAL;
        DECLARE dt$p            POINTER,
                excep$p         POINTER;
        END;
```

```
DQ$OPEN:    PROCEDURE (conn, access, num$buf, excep$p) EXTERNAL;
        DECLARE conn            CONNECTION,
                access          BYTE,
                num$buf         BYTE,
                excep$p         POINTER;
        END;
```

```
DQ$OVERLAY:    PROCEDURE (name$p, excep$p) EXTERNAL;
        DECLARE name$p          POINTER,
                excep$p         POINTER;
        END;
```

```
DQ$READ:    PROCEDURE (conn, buf$p, count, excep$p) WORD EXTERNAL;
        DECLARE conn            CONNECTION,
                buf$p           POINTER,
                count           WORD,
                excep$p         POINTER;
        END;
```

```
DQ$RENAME:    PROCEDURE (old$p, new$p excep$p) EXTERNAL;
        DECLARE old$p           POINTER,
                new$p           POINTER,
                excep$p         POINTER;
        END;
```

```
DQ$RESERVE$IO$MEMORY:    PROCEDURE (number$files, number$buffers, excep$p)
                         EXTERNAL;
        DECLARE number$files    WORD,
                number$buffers  POINTER;
        END;
```

```
DQ$SEEK:    PROCEDURE (conn, mode, offset, excep$p) EXTERNAL;
        DECLARE conn            CONNECTION,
                mode            BYTE,
                offset          DWORD,
                excep$p         POINTER;
        END;
```

```
DQ$SPECIAL:    PROCEDURE (type, parameter$p, excep$p) EXTERNAL;
        DECLARE type            BYTE,
                parameter$p     POINTER,
                excep$p         POINTER;
        END;
```

```
DQ$SWITCH$BUFFER:    PROCEDURE (buffer$p, excep$p) WORD EXTERNAL;
        DECLARE buffer$p        POINTER,
                excep$p         POINTER;
        END;
```

Table B-1. Alphabetical List of Series-IV Service Routines (Cont'd.)

```
DQ$TRAP$CC:    PROCEDURE (handler$p, excep$p) EXTERNAL;
        DECLARE handler$p        POINTER,
                excep$p          POINTER;
        END;
```

```
DQ$TRAP$EXCEPTION:    PROCEDURE (handler$p, excep$p) EXTERNAL;
        DECLARE handler$p        POINTER,
                excep$p          POINTER;
        END;
```

```
DQ$TRUNCATE:    PROCEDURE (conn, excep$p) EXTERNAL;
        DECLARE conn             WORD,
                excep$p          POINTER;
        END;
```

```
DQ$WRITE:    PROCEDURE (conn, buf$p, count, excep$p) EXTERNAL;
        DECLARE conn             CONNECTION,
                buf$p            POINTER,
                count            WORD,
                excep$p          POINTER;
        END;
```

Table B-2. Alphabetical Parameter Definitions

Parameter Name	Routines Using This Parameter and Brief Definition of Parameter
access	OPEN, SEEK, CHANGE$ACCESS A number telling how you plan to use the file, e.g., read, write, or both
arg$p	GET$ARGUMENT Pointer to the 81-byte area you have declared to receive the argument from a line-edited input source
buf$p	READ, WRITE, SWITCH$BUFFER Pointer to the area you have declared for reading from (or writing to) a file; for read or write, it should be at least COUNT bytes long or unintended results will occur
class	CHANGE$ACCESS Specifies the class of users whose access rights are to be changed (0=owner, 1=world, 2-255=reserved)
compl$cod	EXIT A word telling the success of program completion and termination
conn	DETACH, GET$CONNECTION$STATUS, OPEN, SEEK, READ, WRITE, TRUNCATE, CLOSE, SPECIAL, FILE$INFO Connection to a file or device, established earlier via attach or create
count	READ, WRITE The number of bytes you want read or written
delim	GET$ARGUMENT A byte filled by the routine with the delimiter ending the current argument
dt$p	GET$TIME, DECODE$TIME Pointer to the structure you set up for date and time
excep$p	ALL ROUTINES BUT EXIT Pointer to the word you have declared to receive the exception value

Table B-2. Alphabetical Parameter Definitions (Cont'd.)

Parameter Name	Routines Using This Parameter and Brief Definition of Parameter
exception$cod	DECODE$EXCEPTION A word into which you have placed an exception code
exception$p	DECODE$EXCEPTION Pointer to the 81-byte area you have declared for receiving the formatted message corresponding to excep$cod
extension$p	CHANGE$EXTENSION Pointer to the extension as you wish it to be
file$info$p	FILE$INFO Pointer to table used for output
file$ptr	Same as offset
handler$p	TRAP$EXCEPTION, GET$EXCEPTION$HANDLER, TRAP$CC Pointer to the entry-point of your routine to handle current exceptions (or Control C)
id$p	GET$SYSTEM$ID Pointer to the location where you want the system-name or sign-on put
info$p	GET$CONNECTION$STATUS Pointer to the pathname whose connection you need to know
mode	SEEK, FILE$INFO Value representing direction and type of seek operation
name$p	OVERLAY Pointer to the pathname of the overlay to be loaded next
new$p	RENAME Pointer to the pathname as you wish to have it
num$buf	OPEN Number of buffers to be used for I/O to file being opened
number$files	RESERVE$IO$MEMORY Maximum number of files that will be attached at any one time.
number$buffers	RESERVE$IO$MEMORY Maximum number of buffers that will be requested.
offset	SEEK A four-byte unsigned integer representing the number of bytes to move to the file pointer
old$p	RENAME Pointer to the pathname as it is now
path$p	CREATE, DELETE, ATTACH, CHANGE$EXTENSION, CHANGE$ACCESS Pointer to the pathname you wish to use
segbase	FREE, GET$SIZE The word containing the base of a block of bytes
size	ALLOCATE The number of bytes you want to use
type	SPECIAL A value determining whether console input should be line-edited or transparent

APPENDIX C ERROR MESSAGES AND EXCEPTION CODES

iNDX-Series IV Error Messages

A warning does not cause the command to be aborted.

BACKGROUND JOB ACTIVE

Background job is active at the time of user logoff. The logoff is delayed until the background job finshes.

Following is a set of error messages; when these occur, the current command is aborted.

INVALID USER

The user name is not recognized by the system.

INVALID PASSWORD

The password given by the user is incorrect.

SYNTAX ERROR

Illegal syntax was detected.

FILE NOT FOUND

File specified in the command cannot be found.

INVALID CONDITION

Condition specified in the IF, WHILE, UNTIL commands is not valid.

ILLEGAL COMMAND

May be one of a number of conditions:

ORIF command is not preceded by an IF command in the command file.
ELSE command is not preceded by an IF command in the command file.
UNTIL command is not preceded by a REPEAT or COUNT command.
WHILE command is not preceded by a REPEAT or COUNT command.
END command without a matching IF, REPEAT, COUNT command.
Attempt to read without an OPEN command.

INVALID NUMBER

The number specified in the COUNT command is not valid.

THE BACKGROUND IS ALREADY ACTIVE

The BACKGROUND command has been executed while a job is running in the background.

QUEUE DOES NOT EXIST

The queue name specified in the SYSTAT command does not exist.

OS-Series IV Exception Codes

General Exceptions

EXCEPTION 0000H:	SUCCESSFUL COMPLETION
EXCEPTION 0002H:	INSUFFICIENT MEMORY
EXCEPTION 0020H:	FILE ALREADY EXISTS
EXCEPTION 0021H:	FILE DOES NOT EXIST
EXCEPTION 0023H:	UNSUPPORTED OPERATION OR DISK FORMAT
EXCEPTION 0026H:	INSUFFICIENT ACCESS RIGHTS
EXCEPTION 0028H:	SHARED STATE OF FILE PROHIBITS OPEN
EXCEPTION 0031H:	ILLEGAL DEVICE NUMBER
EXCEPTION 0032H:	ALL FNODES ARE IN USE
EXCEPTION 0033H:	NO MORE AVAILABLE SPACE ON DISK
EXCEPTION 0035H:	NON-EMPTY DIRECTORY CANNOT BE DELETED
EXCEPTION 0036H:	DELETE IS PENDING ON THE REQUESTED FILE
EXCEPTION 0037H:	REQUESTED DEVICE IS ALREADY MOUNTED
EXCEPTION 0038H:	REQUESTED DEVICE IS NOT MOUNTED
EXCEPTION 0045H:	DIRECTORY OPERATION ATTEMPTED ON DATA FILE
EXCEPTION 0046H:	FILE REQUESTED IS ALREADY ATTACHED
EXCEPTION 0047H:	FILE SYSTEM ATTACH TABLE IS FULL
EXCEPTION 004AH:	FILE NOT DELETABLE (SYSTEM FILE OR CONTAINS BAD BLOCKS)
EXCEPTION 004BH:	FILE CANNOT BE EXTENDED
EXCEPTION 0081H:	STRING TOO LONG
EXCEPTION 0105H:	INVALID ATTACH NUMBER PASSED AS A PARAMETER
EXCEPTION 0100H:	INVALID PATHNAME—COMPONENT EXCEEDS 14 CHARACTERS

Human Interface/UDI Layer Exceptions

EXCEPTION 2002H:	CALL NOT VALID IN THIS CONTEXT
EXCEPTION 2006H:	INVALID PARAMETER
EXCEPTION 2007H:	INVALID PATHNAME
EXCEPTION 2008H:	TOKEN DOES NOT POINT TO A VALID SEGMENT
EXCEPTION 2009H:	WRONG JOB TYPE
EXCEPTION 200AH:	SYNTAX ERROR
EXCEPTION 201AH:	INVALID TIME
EXCEPTION 201BH:	INVALID DATE
EXCEPTION 201EH:	ALREADY EXISTS
EXCEPTION 201FH:	LIMIT EXCEEDED
EXCEPTION 2050H:	E_COMM
EXCEPTION 2052H:	E_TIMEOUT
EXCEPTION 2054H:	E_REMOTE_ABORT
EXCEPTION 2060H:	INVALID SYSTEM CALL AT
EXCEPTION 2061H:	DIRECTORY FORMAT ERROR DURING WILDCARD PROCESSING
EXCEPTION 2101H:	OPERATION CONFLICT (FILE OPENED FOR WRITE)
EXCEPTION 2102H:	OPERATION CONFLICT (FILE OPENED FOR READ)
EXCEPTION 2201H:	OVERLAY INITIALIZATION FAILED
EXCEPTION 2202H:	INVALID OVERLAY AREA REQUESTED
EXCEPTION 2203H:	OVERLAY INITIALIZATION HAS NOT BEEN COMPLETED

Loader Errors

EXCEPTION 2300H:	UNEXPECTED END OF FILE DURING LOAD
EXCEPTION 2301H:	INVALID LOAD FILE
EXCEPTION 2302:	ILLEGAL LOAD RECORD
EXCEPTION 2303H:	LOAD FILE CONTAINS ABSOLUTE LOAD ADDRESS
EXCEPTION 2304H:	INVALID OVERLAY NAME REQUESTED
EXCEPTION 2305H:	SPECIFIED OVERLAY NOT FOUND
EXCEPTION 2306H:	INVALID LOAD FILE: DATA RECORD EXPECTED
EXCEPTION 2307H:	INVALID LOAD FILE: NOT RELOCATABLE
EXCEPTION 2308H:	LOADER LIMIT EXCEEDED: TOO MANY SEGMENTS
EXCEPTION 2309H:	LOADER LIMIT EXCEEDED: TOO MANY GROUPS
EXCEPTION 230AH:	LOADER LIMIT EXCEEDED: TOO MANY OVERLAYS
EXCEPTION 230BH:	NON-RELOCATABLE SEGMENT ENCOUNTERED
EXCEPTION 230CH:	INVALID LOAD FILE: BAD SEGMENT DEFINITION RECORD
EXCEPTION 230DH:	NON-RELOCATABLE GROUP ENCOUNTERED
EXCEPTION 230EH:	INVALID LOAD FILE: BAD SEGMENT IN GROUP DEF RECORD
EXCEPTION 230FH:	INVALID LOAD FILE: BAD GROUP DEFINITION RECORD
EXCEPTION 2310H:	INVALID LOAD FILE: OVERLAY DEFINITION RECORD EXPECTED
EXCEPTION 2311H:	INVALID LOAD FILE: ILLEGAL LOAD TIME FIXUP
EXCEPTION 2321H:	INVALID LOAD FILE: INVALID REGISTER INITIALIZATION RECORD
EXCEPTION 2322H:	LOAD FILE CONTAINS UNRESOLVED EXTERNAL REFERENCES
EXCEPTION 2323H:	LOAD FILE HAS BAD REGISTER INIT, UNRESOLVED EXTERNALS
EXCEPTION 2324H:	INITIAL STACK FOR LOADED FILE IS TOO SMALL FOR STARTUP
EXCEPTION 2325H:	LOAD FILE HAS BAD REGISTER INIT AND STACK IS TOO SMALL
EXCEPTION 2326H:	LOAD FILE HAS UNRESOLVED EXTERNALS AND STACK IS TOO SMALL
EXCEPTION 2327H:	LOAD FILE HAS BAD REGISTER INIT, UNRESOLVEDS, STACK TOO SMALL

Job Control Exceptions

EXCEPTION 2400H:	QUEUE LIMIT EXCEEDED
EXCEPTION 2401H:	SERVER LIMIT EXCEEDED
EXCEPTION 2402H:	QUEUE FULL
EXCEPTION 2403H:	SERVER DOES NOT EXIST
EXCEPTION 2405H:	QUEUE DOES NOT EXIST
EXCEPTION 2406H:	JOB(S) NOT FOUND
EXCEPTION 2407H:	USER
EXCEPTION 2408H:	SYSTEM FILES
EXCEPTION 2409H:	JOB NUMBER IS INCONSISTENT WITH SPECIFIED QUEUE
EXCEPTION 240AH:	DJC ERROR: E_BUFFER_TOO_SMALL
EXCEPTION 240BH:	DJC ERROR: E_INVALID_DJC_REQUEST

EXCEPTION 240CH:	DJC ERROR: E_TIME
EXCEPTION 240EH:	DJC ERROR: E_BREAK_OCCURRED
EXCEPTION 2410H:	DJC: END IMPORT
EXCEPTION 2411H:	DJC ERROR: E_DJC_BAD_ACK
EXCEPTION 2412H:	DJC ERROR: E_DJC_COMM_SYNCH
EXCEPTION 2413H:	DJC ERROR: E_DJC_JOB_NOT_EXECUTING
EXCEPTION 2414H:	DJC CONSISTENCY ERROR: QUEUE TABLE FILE INVALID
EXCEPTION 2602H:	DJC TCL ERROR: INVALID REQUEST
EXCEPTION 2604H:	DJC TCL ERROR: NO RESOURCE AVAILABLE
EXCEPTION 2608H:	DJC ERROR: E_BUFF_TOO_SHORT
EXCEPTION 260AH:	DJC TCL ERROR: TRIED TO SEND AFTER CONNECTION WAS CLOSED
EXCEPTION 260CH:	DJC TCL ERROR: LOCAL CLIENT OF TCL ISSUED ABORT
EXCEPTION 260EH:	DJC TCL ERROR: REMOTE CLIENT OF TCL ISSUED ABORT
EXCEPTION 2610H:	DJC TCL ERROR: LOCAL ABORT TIMEOUT
EXCEPTION 2612H:	DJC: CLOSE COMPLETE (NOT AN ERROR)
EXCEPTION 2614H:	DJC TCL ERROR: INVALID REQUEST BLOCK POINTER
EXCEPTION 2616H:	DJC PROTOCOL ERROR: PRB TO CLOSED CONNECTION

BIOS Device Specific Exceptions

For errors in the range 3000H to 380FH, the low order nibble (half byte) is a number that indicates the device on which the exception occurred. For different values of the device number, a different message table is used. The error code is of the form 3*xxn*H. If *n* = 0 or 1, it is a flippy error. If *n* = 2, 3, 4 or 5, the error occurred on a 5440 hard disk. If *n* = 6 or above, the error occurred on a Winchester type hard disk.

Messages for Device Errors on 5-1/4″ Flexible Disks

If *n* = 0, 'dev' = 'FL0' (flippy drive zero)
If *n* = 1, 'dev' = 'FL1' (flippy drive one)

Device Specific Exceptions for 5440 Hard Disk

If *n* = 2, then 'dev' = 'HD0' (fixed platter of first 5440)
If *n* = 3, then 'dev' = 'HD1' (removable platter of first 5440)
If *n* = 4, then 'dev' = 'HD2' (fixed platter of second 5440)
If *n* = 5, then 'dev' = 'HD3' (removable platter of second 5440)

Device Specific Exceptions for Winchester Disks

If *n* = 6, then 'dev' = 'WD0' (first device)
If *n* = 7, then 'dev' = 'WD1' (second device)
If *n* = 8, then 'dev' = 'WD2' (third device)
If *n* = 9, then 'dev' = 'WD3' (fourth device)

If *n* = a, then 'dev' = 'WF0' (first device)
If *n* = b, then 'dev' = 'WF1' (second device)
If *n* = c, then 'dev' = 'WF2' (third device)
If *n* = d, then 'dev' = 'WF3' (fourth device)

If *n* = e, then 'dev' = 'WM0' (first device)

EXCEPTION 3010H:	FL0 ERR, UNEXPECTED DELETED DATA ADDRESS MARK ON DISK
EXCEPTION 3011H:	FL1 ERR, UNEXPECTED DELETED DATA ADDRESS MARK ON DISK
EXCEPTION 3012H:	HD0 ERR, HARDWARE OR DISK FAIL, BAD ID FIELD CONTENTS
EXCEPTION 3013H:	HD1 ERR, HARDWARE OR DISK FAIL, BAD ID FIELD CONTENTS
EXCEPTION 3014H:	HD2 ERR, HARDWARE OR DISK FAIL, BAD ID FIELD CONTENTS
EXCEPTION 3015H:	HD3 ERR, HARDWARE OR DISK FAIL, BAD ID FIELD CONTENTS
EXCEPTION 3020H:	FL0 ERR, HARDWARE OR DISK FAIL, BAD DATA FIELD CRC
EXCEPTION 3021H:	FL1 ERR, HARDWARE OR DISK FAIL, BAD DATA FIELD CRC
EXCEPTION 3022H:	HD0 ERR, HARDWARE OR DISK FAIL, BAD DATA FIELD CRC
EXCEPTION 3023H:	HD1 ERR, HARDWARE OR DISK FAIL, BAD DATA FIELD CRC
EXCEPTION 3024H:	HD2 ERR, HARDWARE OR DISK FAIL, BAD DATA FIELD CRC
EXCEPTION 3025H:	HD3 ERR, HARDWARE OR DISK FAIL, BAD DATA FIELD CRC
EXCEPTION 3026H:	WD0 ERR, HARDWARE OR DISK FAIL, BAD DATA FIELD CRC
EXCEPTION 3027H:	DEV ERR, HARDWARE OR DISK FAIL, BAD DATA FIELD CRC
EXCEPTION 3028H:	DEV ERR, HARDWARE OR DISK FAIL, BAD DATA FIELD ECC
EXCEPTION 3029H:	DEV ERR, HARDWARE OR DISK FAIL, BAD DATA FIELD ECC
EXCEPTION 302AH:	DEV ERR, HARDWARE OR DISK FAIL, BAD DATA FIELD ECC
EXCEPTION 302BH:	DEV ERR, HARDWARE OR DISK FAIL, BAD DATA FIELD ECC
EXCEPTION 302CH:	DEV ERR, HARDWARE OR DISK FAIL, BAD DATA FIELD ECC
EXCEPTION 302DH:	DEV ERR, HARDWARE OR DISK FAIL, BAD DATA FIELD ECC
EXCEPTION 302EH:	DEV ERR, HARDWARE OR DISK FAIL, BAD DATA FIELD ECC
EXCEPTION 302FH:	DEV ERR, HARDWARE OR DISK FAIL, BAD DATA FIELD ECC
EXCEPTION 3040H:	FL0 ERR, HARDWARE FAIL, ID CYC FIELD SEEK CYL
EXCEPTION 3041H:	FL1 ERR, HARDWARE FAIL, ID CYC FIELD SEEK CYL
EXCEPTION 3042H:	HD0 ERR, HARDWARE OR DRIVE FAIL, SEEK ERROR
EXCEPTION 3043:	HD1 ERR, HARDWARE OR DRIVE FAIL, SEEK ERROR
EXCEPTION 3044H:	HD2 ERR, HARDWARE OR DRIVE FAIL, SEEK ERROR
EXCEPTION 3045H:	HD3 ERR, HARDWARE OR DRIVE FAIL, SEEK ERROR
EXCEPTION 3046H:	WD0 ERR, HARDWARE OR DRIVE FAIL, SEEK ERROR

EXCEPTION 3047H:	DEV ERR, HARDWARE OR DRIVE FAIL, SEEK ERROR
EXCEPTION 3048H:	DEV ERR, HARDWARE OR DRIVE FAIL, SEEK ERROR
EXCEPTION 3049H:	DEV ERR, HARDWARE OR DRIVE FAIL, SEEK ERROR
EXCEPTION 304AH:	DEV ERR, HARDWARE OR DRIVE FAIL, SEEK ERROR
EXCEPTION 304BH:	DEV ERR, HARDWARE OR DRIVE FAIL, SEEK ERROR
EXCEPTION 304CH:	DEV ERR, HARDWARE OR DRIVE FAIL, SEEK ERROR
EXCEPTION 304DH:	DEV ERR, HARDWARE OR DRIVE FAIL, SEEK ERROR
EXCEPTION 304EH:	DEV ERR, HARDWARE OR DRIVE FAIL, SEEK ERROR
EXCEPTION 304FH:	DEV ERR, HARDWARE OR DRIVE FAIL, SEEK ERROR
EXCEPTION 3080H:	FL0 ERR, OS ERR, ATTEMPTING TO ACCESS NON_EXISTENT SECTOR
EXCEPTION 3081H:	FL1 ERR, OS ERR, ATTEMPTING TO ACCESS NON_EXISTENT SECTOR
EXCEPTION 3082H:	HD0 ERR, OS ERROR, ATTEMPTING TO ACCESS NON_EXISTENT SECTOR
EXCEPTION 3083H:	HD1 ERR, OS ERROR, ATTEMPTING TO ACCESS NON_EXISTENT SECTOR
EXCEPTION 3084H:	HD2 ERR, OS ERROR, ATTEMPTING TO ACCESS NON_EXISTENT SECTOR
EXCEPTION 3085H:	HD3 ERR, OS ERROR, ATTEMPTING TO ACCESS NON_EXISTENT SECTOR
EXCEPTION 30A0H:	FL0 ERR, HARDWARE OR DISK FAIL, BAD ID FIELD CRC
EXCEPTION 30A1H:	FL1 ERR, HARDWARE OR DISK FAIL, BAD ID FIELD CRC
EXCEPTION 30A2H:	HD0 ERR, HARDWARE OR DISK FAIL, BAD ID FIELD CRC CODE
EXCEPTION 30A3H:	HD1 ERR, HARDWARE OR DISK FAIL, BAD ID FIELD CRC CODE
EXCEPTION 30A4H:	HD2 ERR, HARDWARE OR DISK FAIL, BAD ID FIELD CRC CODE
EXCEPTION 30A5H:	HD3 ERR, HARDWARE OR DISK FAIL, BAD ID FIELD CRC CODE
EXCEPTION 30A6H:	WD0 ERR, HARDWARE OR DISK FAIL, BAD ID FIELD ECC
EXCEPTION 30A7H:	DEV ERR, HARDWARE OR DISK FAIL, BAD ID FIELD ECC
EXCEPTION 30A8H:	DEV ERR, HARDWARE OR DISK FAIL, BAD ID FIELD ECC
EXCEPTION 30A9H:	DEV ERR, HARDWARE OR DISK FAIL, BAD ID FIELD ECC
EXCEPTION 30AAH:	DEV ERR, HARDWARE OR DISK FAIL, BAD ID FIELD ECC
EXCEPTION 30ABH:	DEV ERR, HARDWARE OR DISK FAIL, BAD ID FIELD ECC
EXCEPTION 30ACH:	DEV ERR, HARDWARE OR DISK FAIL, BAD ID FIELD ECC
EXCEPTION 30ADH:	DEV ERR, HARDWARE OR DISK FAIL, BAD ID FIELD ECC

EXCEPTION 30AEH:	DEV ERR, HARDWARE OR DISK FAIL, BAD ID FIELD ECC
EXCEPTION 30AFH:	DEV ERR, HARDWARE OR DISK FAIL, BAD ID FIELD ECC
EXCEPTION 30B2H:	HD0 ERR, OS OR CONTROLLER FAIL, BAD PROTOCOL
EXCEPTION 30B3H:	HD1 ERR, OS OR CONTROLLER FAIL, BAD PROTOCOL
EXCEPTION 30B4H:	HD2 ERR, OS OR CONTROLLER FAIL, BAD PROTOCOL
EXCEPTION 30B5H:	HD3 ERR, OS OR CONTROLLER FAIL, BAD PROTOCOL
EXCEPTION 30C2H:	HD0 ERR, OS OR DRIVE FAIL, CYLINDER ADDRESS OF OF BOUNDS
EXCEPTION 30C3H:	HD1 ERR, OS OR DRIVE FAIL, CYLINDER ADDRESS OF OF BOUNDS
EXCEPTION 30C4H:	HD2 ERR, OS OR DRIVE FAIL, CYLINDER ADDRESS OF OF BOUNDS
EXCEPTION 30C5H:	HD3 ERR, OS OR DRIVE FAIL, CYLINDER ADDRESS OF OF BOUNDS
EXCEPTION 30E0H:	FL0 ERR, HARDWARE OR DISK FAIL, NO ID ADDRESS MARK
EXCEPTION 30E1H:	FL1 ERR, HARDWARE OR DISK FAIL, NO ID ADDRESS MARK
EXCEPTION 30E2H:	HD0 ERR, DISK BAD, CAN'T FIND SECTOR
EXCEPTION 30E3H:	HD1 ERR, DISK BAD, CAN'T FIND SECTOR
EXCEPTION 30E4H:	HD2 ERR, DISK BAD, CAN'T FIND SECTOR
EXCEPTION 30E5H:	HD3 ERR, DISK BAD, CAN'T FIND SECTOR
EXCEPTION 30E6H:	WD0 ERR, DISK BAD, CAN'T FIND SECTOR
EXCEPTION 30E7H:	DEV ERR, DISK BAD, CAN'T FIND SECTOR
EXCEPTION 30E8H:	DEV ERR, DISK BAD, CAN'T FIND SECTOR
EXCEPTION 30E9H:	DEV ERR, DISK BAD, CAN'T FIND SECTOR
EXCEPTION 30EAH:	DEV ERR, DISK BAD, CAN'T FIND SECTOR
EXCEPTION 30EBH:	DEV ERR, DISK BAD, CAN'T FIND SECTOR
EXCEPTION 30ECH:	DEV ERR, DISK BAD, CAN'T FIND SECTOR
EXCEPTION 30EDH:	DEV ERR, DISK BAD, CAN'T FIND SECTOR
EXCEPTION 30EEH:	DEV ERR, DISK BAD, CAN'T FIND SECTOR
EXCEPTION 30EFH:	DEV ERR, DISK BAD, CAN'T FIND SECTOR
EXCEPTION 30F0H:	FL0 ERR, HARDWARE OR DISK FAIL, NO DATA ADDRESS MARK
EXCEPTION 30F1H:	FL1 ERR, HARDWARE OR DISK FAIL, NO DATA ADDRESS MARK
EXCEPTION 30F2H:	HD0 ERR, CONTROLLER OR DISK FAIL, NO BEGIN OF DATA FIELD MARK
EXCEPTION 30F3H:	HD1 ERR, CONTROLLER OR DISK FAIL, NO BEGIN OF DATA FIELD MARK
EXCEPTION 30F4H:	HD2 ERR, CONTROLLER OR DISK FAIL, NO BEGIN OF DATA FIELD MARK
EXCEPTION 30F5H:	HD3 ERR, CONTROLLER OR DISK FAIL, NO BEGIN OF DATA FIELD MARK
EXCEPTION 3100H:	FL0 ERR, MULTIBUS CONTENTION, ACCESS OVER/UNDER RUN
EXCEPTION 3101H:	FL1 ERR, MULTIBUS CONTENTION, ACCESS OVER/UNDER RUN
EXCEPTION 3102H:	HD0 ERR, MULTIBUS CONTENTION, ACCESS OVER/UNDER RUN
EXCEPTION 3103H:	HD1 ERR, MULTIBUS CONTENTION, ACCESS OVER/UNDER RUN

EXCEPTION 3104H:	HD2 ERR, MULTIBUS CONTENTION, ACCESS OVER/UNDER RUN
EXCEPTION 3105H:	HD3 ERR, MULTIBUS CONTENTION, ACCESS OVER/UNDER RUN
EXCEPTION 3116H:	WD0 ERR, CONTROLLER RAM TEST FAIL
EXCEPTION 3117H:	WD1 ERR, CONTROLLER RAM TEST FAIL
EXCEPTION 3118H:	WD2 ERR, CONTROLLER RAM TEST FAIL
EXCEPTION 3119H:	WD3 ERR, CONTROLLER RAM TEST FAIL
EXCEPTION 311AH:	WF0 ERR, CONTROLLER RAM TEST FAIL
EXCEPTION 311BH:	WF1 ERR, CONTROLLER RAM TEST FAIL
EXCEPTION 311CH:	WF2 ERR, CONTROLLER RAM TEST FAIL
EXCEPTION 311DH:	WF3 ERR, CONTROLLER RAM TEST FAIL
EXCEPTION 311EH:	WM0 ERR, CONTROLLER RAM TEST FAIL
EXCEPTION 311FH:	WM1 ERR, CONTROLLER RAM TEST FAIL
EXCEPTION 3126H:	WD0 ERR, CONTROLLER PROM TEST FAIL
EXCEPTION 3127H:	WD1 ERR, CONTROLLER PROM TEST FAIL
EXCEPTION 3128H:	WD2 ERR, CONTROLLER PROM TEST FAIL
EXCEPTION 3129H:	WD3 ERR, CONTROLLER PROM TEST FAIL
EXCEPTION 312AH:	WF0 ERR, CONTROLLER PROM TEST FAIL
EXCEPTION 312BH:	WF1 ERR, CONTROLLER PROM TEST FAIL
EXCEPTION 312CH:	WF2 ERR, CONTROLLER PROM TEST FAIL
EXCEPTION 312DH:	WF3 ERR, CONTROLLER PROM TEST FAIL
EXCEPTION 312EH:	WM0 ERR, CONTROLLER PROM TEST FAIL
EXCEPTION 312FH:	WM1 ERR, CONTROLLER PROM TEST FAIL
EXCEPTION 3136H:	WD0 ERR, OS OR CONTROLLER FAIL, SEEK IN PROGRESS
EXCEPTION 3137H:	WD1 ERR, OS OR CONTROLLER FAIL, SEEK IN PROGRESS
EXCEPTION 3138H:	WD2 ERR, OS OR CONTROLLER FAIL, SEEK IN PROGRESS
EXCEPTION 3139H:	WD3 ERR, OS OR CONTROLLER FAIL, SEEK IN PROGRESS
EXCEPTION 313AH:	WF0 ERR, OS OR CONTROLLER FAIL, SEEK IN PROGRESS
EXCEPTION 313BH:	WF1 ERR, OS OR CONTROLLER FAIL, SEEK IN PROGRESS
EXCEPTION 313CH:	WF2 ERR, OS OR CONTROLLER FAIL, SEEK IN PROGRESS
EXCEPTION 312DH:	WF3 ERR, OS OR CONTROLLER FAIL, SEEK IN PROGRESS
EXCEPTION 313EH:	WM0 ERR, OS OR CONTROLLER FAIL, SEEK IN PROGRESS
EXCEPTION 313FH:	WM1 ERR, OS OR CONTROLLER FAIL, SEEK IN PROGRESS
EXCEPTION 3146H:	WD0 ERR, OS OR CONTROLLER FAIL, TRACK TYPE DISALLOWS OPERATION
EXCEPTION 3147H:	WD1 ERR, OS OR CONTROLLER FAIL, TRACK TYPE DISALLOWS OPERATION
EXCEPTION 3148H:	WD2 ERR, OS OR CONTROLLER FAIL, TRACK TYPE DISALLOWS OPERATION
EXCEPTION 3149H:	WD3 ERR, OS OR CONTROLLER FAIL, TRACK TYPE DISALLOWS OPERATION
EXCEPTION 314AH:	WF0 ERR, OS OR CONTROLLER FAIL, TRACK TYPE DISALLOWS OPERATION
EXCEPTION 3148H:	WF1 ERR, OS OR CONTROLLER FAIL, TRACK TYPE DISALLOWS OPERATION
EXCEPTION 314CH:	WF2 ERR, OS OR CONTROLLER FAIL, TRACK TYPE DISALLOWS OPERATION

EXCEPTION 314DH:	WF3 ERR, OS OR CONTROLLER FAIL, TRACK TYPE DISALLOWS OPERATION
EXCEPTION 314EH:	WM0 ERR, OS OR CONTROLLER FAIL, TRACK TYPE DISALLOWS OPERATION
EXCEPTION 314FH:	WM1 ERR, OS OR CONTROLLER FAIL, TRACK TYPE DISALLOWS OPERATION
EXCEPTION 3156H:	WD0 ERR, OS OR CONTROLLER FAIL, ACCESS BEYOND END OF MEDIA
EXCEPTION 3157H:	WD1 ERR, OS OR CONTROLLER FAIL, ACCESS BEYOND END OF MEDIA
EXCEPTION 3158H:	WD2 ERR, OS OR CONTROLLER FAIL, ACCESS BEYOND END OF MEDIA
EXCEPTION 3159H:	WD3 ERR, OS OR CONTROLLER FAIL, ACCESS BEYOND END OF MEDIA
EXCEPTION 315AH:	WF0 ERR, OS OR CONTROLLER FAIL, ACCESS BEYOND END OF MEDIA
EXCEPTION 315BH:	WF1 ERR, OS OR CONTROLLER FAIL, ACCESS BEYOND END OF MEDIA
EXCEPTION 315CH:	WF2 ERR, OS OR CONTROLLER FAIL, ACCESS BEYOND END OF MEDIA
EXCEPTION 315DH:	WF3 ERR, OS OR CONTROLLER FAIL, ACCESS BEYOND END OF MEDIA
EXCEPTION 315EH:	WM0 ERR, OS OR CONTROLLER FAIL, ACCESS BEYOND END OF MEDIA
EXCEPTION 315FH:	WM1 ERR, OS OR CONTROLLER FAIL, ACCESS BEYOND END OF MEDIA
EXCEPTION 3166H:	WD0 ERR, DISK BAD, INCORRECT SECTOR SIZE FOUND
EXCEPTION 3167H:	WD1 ERR, DISK BAD, INCORRECT SECTOR SIZE FOUND
EXCEPTION 3168H:	WD2 ERR, DISK BAD, INCORRECT SECTOR SIZE FOUND
EXCEPTION 3169H:	WD3 ERR, DISK BAD, INCORRECT SECTOR SIZE FOUND
EXCEPTION 316AH:	WF0 ERR, DISK BAD, INCORRECT SECTOR SIZE FOUND
EXCEPTION 316BH:	WF1 ERR, DISK BAD, INCORRECT SECTOR SIZE FOUND
EXCEPTION 316CH:	WF2 ERR, DISK BAD, INCORRECT SECTOR SIZE FOUND
EXCEPTION 316DH:	WF3 ERR, DISK BAD, INCORRECT SECTOR SIZE FOUND
EXCEPTION 316EH:	WM0 ERR, DISK BAD, INCORRECT SECTOR SIZE FOUND
EXCEPTION 316FH:	WM1 ERR, DISK BAD, INCORRECT SECTOR SIZE FOUND
EXCEPTION 3176H:	WD0 ERR, CONTROLLER DIAGNOSTIC FAULT
EXCEPTION 3177H:	WD1 ERR, CONTROLLER DIAGNOSTIC FAULT
EXCEPTION 3178H:	WD2 ERR, CONTROLLER DIAGNOSTIC FAULT
EXCEPTION 3179H:	WD3 ERR, CONTROLLER DIAGNOSTIC FAULT
EXCEPTION 317AH:	WF0 ERR, CONTROLLER DIAGNOSTIC FAULT
EXCEPTION 317BH:	WF1 ERR, CONTROLLER DIAGNOSTIC FAULT
EXCEPTION 317CH:	WF2 ERR, CONTROLLER DIAGNOSTIC FAULT
EXCEPTION 317DH:	WF3 ERR, CONTROLLER DIAGNOSTIC FAULT
EXCEPTION 317EH:	WM0 ERR, CONTROLLER DIAGNOSTIC FAULT
EXCEPTION 317FH:	WM1 ERR, CONTROLLER DIAGNOSTIC FAULT
EXCEPTION 3186H:	WD0 ERR, CONTROLLER OR DRIVE FAIL, NO INDEX SIGNAL

EXCEPTION 3187H:	WD1 ERR, CONTROLLER OR DRIVE FAIL, NO INDEX SIGNAL
EXCEPTION 3188H:	WD2 ERR, CONTROLLER OR DRIVE FAIL, NO INDEX SIGNAL
EXCEPTION 3189H:	WD3 ERR, CONTROLLER OR DRIVE FAIL, NO INDEX SIGNAL
EXCEPTION 318AH:	WF0 ERR, CONTROLLER OR DRIVE FAIL, NO INDEX SIGNAL
EXCEPTION 318BH:	WF1 ERR, CONTROLLER OR DRIVE FAIL, NO INDEX SIGNAL
EXCEPTION 318CH:	WF2 ERR, CONTROLLER OR DRIVE FAIL, NO INDEX SIGNAL
EXCEPTION 318DH:	WF3 ERR, CONTROLLER OR DRIVE FAIL, NO INDEX SIGNAL
EXCEPTION 318EH:	WM0 ERR, CONTROLLER OR DRIVE FAIL, NO INDEX SIGNAL
EXCEPTION 318FH:	WM1 ERR, CONTROLLER OR DRIVE FAIL, NO INDEX SIGNAL
EXCEPTION 3196H:	WD0 ERR, OS OR CONTROLLER FAIL, INVALID FUNCTION CODE PASSED
EXCEPTION 3197H:	WD1 ERR, OS OR CONTROLLER FAIL, INVALID FUNCTION CODE PASSED
EXCEPTION 3198H:	WD2 ERR, OS OR CONTROLLER FAIL, INVALID FUNCTION CODE PASSED
EXCEPTION 3199H:	WD3 ERR, OS OR CONTROLLER FAIL, INVALID FUNCTION CODE PASSED
EXCEPTION 319AH:	WF0 ERR, OS OR CONTROLLER FAIL, INVALID FUNCTION CODE PASSED
EXCEPTION 319BH:	WF1 ERR, OS OR CONTROLLER FAIL, INVALID FUNCTION CODE PASSED
EXCEPTION 319CH:	WF2 ERR, OS OR CONTROLLER FAIL, INVALID FUNCTION CODE PASSED
EXCEPTION 319DH:	WF3 ERR, OS OR CONTROLLER FAIL, INVALID FUNCTION CODE PASSED
EXCEPTION 319EH:	WM0 ERR, OS OR CONTROLLER FAIL, INVALID FUNCTION CODE PASSED
EXCEPTION 319FH:	WM1 ERR, OS OR CONTROLLER FAIL, INVALID FUNCTION CODE PASSED
EXCEPTION 31A6H:	WD0 ERR, OS OR CONTROLLER FAIL, DISK ADDRESS OUT OF BOUNDS
EXCEPTION 31A7H:	WD1 ERR, OS OR CONTROLLER FAIL, DISK ADDRESS OUT OF BOUNDS
EXCEPTION 31A8H:	WD2 ERR, OS OR CONTROLLER FAIL, DISK ADDRESS OUT OF BOUNDS
EXCEPTION 31A9H:	WD3 ERR, OS OR CONTROLLER FAIL, DISK ADDRESS OUT OF BOUNDS
EXCEPTION 31AAH:	WF0 ERR, OS OR CONTROLLER FAIL, DISK ADDRESS OUT OF BOUNDS
EXCEPTION 31ABH:	WF1 ERR, OS OR CONTROLLER FAIL, DISK ADDRESS OUT OF BOUNDS
EXCEPTION 31ACH:	WF2 ERR, OS OR CONTROLLER FAIL, DISK ADDRESS OUT OF BOUNDS
EXCEPTION 31ADH:	WF3 ERR, OS OR CONTROLLER FAIL, DISK ADDRESS OUT OF BOUNDS
EXCEPTION 31AEH:	WM0 ERR, OS OR CONTROLLER FAIL, DISK ADDRESS OUT OF BOUNDS
EXCEPTION 31AFH:	WM1 ERR, OS OR CONTROLLER FAIL, DISK ADDRESS OUT OF BOUNDS

EXCEPTION 3200H:	FL0 ERR, ATTEMPT TO WRITE WITH HARDWARE WRITE PROTECT SET
EXCEPTION 3201H:	FI1 ERR, ATTEMPT TO WRITE WITH HARDWARE WRITE PROTECT SET
EXCEPTION 3202H:	HD0 ERR, ATTEMPT TO WRITE WITH HARDWARE WRITE PROTECT SET
EXCEPTION 3202H:	HD1 ERR, ATTEMPT TO WRITE WITH HARDWARE WRITE PROTECT SET
EXCEPTION 3204H:	HD2 ERR, ATTEMPT TO WRITE WITH HARDWARE WRITE PROTECT SET
EXCEPTION 3205H:	HD3 ERR, ATTEMPT TO WRITE WITH HARDWARE WRITE PROTECT SET
EXCEPTION 3206H:	WD0 ERR, ATTEMPT TO WRITE WITH HARDWARE WRITE PROTECT SET
EXCEPTION 3207H:	WD1 ERR, ATTEMPT TO WRITE WITH HARDWARE WRITE PROTECT SET
EXCEPTION 3208H:	WD2 ERR, ATTEMPT TO WRITE WITH HARDWARE WRITE PROTECT SET
EXCEPTION 3209H:	WD3 ERR, ATTEMPT TO WRITE WITH HARDWARE WRITE PROTECT SET
EXCEPTION 320AH:	WF0 ERR, ATTEMPT TO WRITE WITH HARDWARE WRITE PROTECT SET
EXCEPTION 320BH:	WF1 ERR, ATTEMPT TO WRITE WITH HARDWARE WRITE PROTECT SET
EXCEPTION 320CH:	WF2 ERR, ATTEMPT TO WRITE WITH HARDWARE WRITE PROTECT SET
EXCEPTION 320DH:	WF3 ERR, ATTEMPT TO WRITE WITH HARDWARE WRITE PROTECT SET
EXCEPTION 320EH:	WM0 ERR, ATTEMPT TO WRITE WITH HARDWARE WRITE PROTECT SET
EXCEPTION 320FH:	WM1 ERR, ATTEMPT TO WRITE WITH HARDWARE WRITE PROTECT SET
EXCEPTION 3400H:	FL0 ERR, DRIVE IS INDICATING A HARDWARE FAILURE
EXCEPTION 3401H:	FI1 ERR, DRIVE IS INDICATING A HARDWARE FAILURE
EXCEPTION 3402H:	HD0 ERR, DRIVE IS INDICATING A HARDWARE FAULT
EXCEPTION 3403H:	HD1 ERR, DRIVE IS INDICATING A HARDWARE FAULT
EXCEPTION 3404H:	HD2 ERR, DRIVE IS INDICATING A HARDWARE FAULT
EXCEPTION 3405H:	HD3 ERR, DRIVE IS INDICATING A HARDWARE FAULT
EXCEPTION 3406H:	WD0 ERR, DRIVE IS INDICATING A HARDWARE FAULT
EXCEPTION 3407H:	WD1 ERR, DRIVE IS INDICATING A HARDWARE FAULT
EXCEPTION 3408H:	WD2 ERR, DRIVE IS INDICATING A HARDWARE FAULT
EXCEPTION 3409H:	WD3 ERR, DRIVE IS INDICATING A HARDWARE FAULT
EXCEPTION 340AH:	WF0 ERR, DRIVE IS INDICATING A HARDWARE FAULT
EXCEPTION 340BH:	WF1 ERR, DRIVE IS INDICATING A HARDWARE FAULT
EXCEPTION 340CH:	WF2 ERR, DRIVE IS INDICATING A HARDWARE FAULT

EXCEPTION 340DH:	WF3 ERR, DRIVE IS INDICATING A HARDWARE FAULT
EXCEPTION 340EH:	WM0 ERR, DRIVE IS INDICATING A HARDWARE FAULT
EXCEPTION 340FH:	WM1 ERR, DRIVE IS INDICATING A HARDWARE FAULT
EXCEPTION 3700H:	FL0 ERR, DISK BAD, CAN'T FIND SECTOR
EXCEPTION 3701H:	FL1 ERR, DISK BAD, CAN'T FIND SECTOR
EXCEPTION 3710H:	FL0 ERR, UNEXPECTED BAD TRACK FLAG ON DISK
EXCEPTION 3711H:	FL1 ERR, UNEXPECTED BAD TRACK FLAG ON DISK
EXCEPTION 3720H:	FL0 ERR, HARDWARE FAIL, SEEK DID NOT GET TO EXPECTED TRACK
EXCEPTION 3721H:	FL1 ERR, HARDWARE FAIL, SEEK DID NOT GET TO EXPECTED TRACK
EXCEPTION 3726H:	WD0 ERR, DRIVE FAIL, ID CYL ADDR DOESN'T MATCH SEEK ADDR
EXCEPTION 3727H:	WD1 ERR, DRIVE FAIL, ID CYL ADDR DOESN'T MATCH SEEK ADDR
EXCEPTION 3728H:	WD2 ERR, DRIVE FAIL, ID CYL ADDR DOESN'T MATCH SEEK ADDR
EXCEPTION 3729H:	WD3 ERR, DRIVE FAIL, ID CYL ADDR DOESN'T MATCH SEEK ADDR
EXCEPTION 372AH:	WF0 ERR, DRIVE FAIL, ID CYL ADDR DOESN'T MATCH SEEK ADDR
EXCEPTION 372BH:	WF1 ERR, DRIVE FAIL, ID CYL ADDR DOESN'T MATCH SEEK ADDR
EXCEPTION 372CH:	WF2 ERR, DRIVE FAIL, ID CYL ADDR DOESN'T MATCH SEEK ADDR
EXCEPTION 372DH:	WF3 ERR, DRIVE FAIL, ID CYL ADDR DOESN'T MATCH SEEK ADDR
EXCEPTION 372EH:	WM0 ERR, DRIVE FAIL, ID CYL ADDR DOESN'T MATCH SEEK ADDR
EXCEPTION 372FH:	WM1 ERR, DRIVE FAIL, ID CYL ADDR DOESN'T MATCH SEEK ADDR
EXCEPTION 3780H:	FL0 ERR, UNKNOWN ERROR CODE FROM CONTROLLER
EXCEPTION 3781H:	FL1 ERR, UNKNOWN ERROR CODE FROM CONTROLLER
EXCEPTION 3786H:	WD0 ERR, UNKNOWN ERROR CODE FROM CONTROLLER
EXCEPTION 3787H:	WD1 ERR, UNKNOWN ERROR CODE FROM CONTROLLER
EXCEPTION 3788H:	WD2 ERR, UNKNOWN ERROR CODE FROM CONTROLLER
EXCEPTION 3789H:	WD3 ERR, UNKNOWN ERROR CODE FROM CONTROLLER
EXCEPTION 378AH:	WF0 ERR, UNKNOWN ERROR CODE FROM CONTROLLER
EXCEPTION 378BH:	WF1 ERR, UNKNOWN ERROR CODE FROM CONTROLLER
EXCEPTION 378CH:	WF2 ERR, UNKNOWN ERROR CODE FROM CONTROLLER
EXCEPTION 378DH:	WF3 ERR, UNKNOWN ERROR CODE FROM CONTROLLER
EXCEPTION 378EH:	WM0 ERR, UNKNOWN ERROR CODE FROM CONTROLLER

EXCEPTION 378FH:	WM1 ERR, UNKNOWN ERROR CODE FROM CONTROLLER
EXCEPTION 3800H:	FL0 ERR, DISK NOT INSERTED AND SPINNING
EXCEPTION 3801H:	FL1 ERR, DISK NOT INSERTED AND SPINNING
EXCEPTION 3802H:	HD0 ERR, DRIVE NOT READY
EXCEPTION 3803H:	HD1 ERR, DRIVE NOT READY
EXCEPTION 3804H:	HD2 ERR, DRIVE NOT READY
EXCEPTION 3805H:	HD3 ERR, DRIVE NOT READY
EXCEPTION 3806H:	WD0 ERR, DRIVE NOT READY
EXCEPTION 3807H:	WD1 ERR, DRIVE NOT READY
EXCEPTION 3808H:	WD2 ERR, DRIVE NOT READY
EXCEPTION 3809H:	WD3 ERR, DRIVE NOT READY
EXCEPTION 380AH:	WF0 ERR, DRIVE NOT READY
EXCEPTION 380BH:	WF1 ERR, DRIVE NOT READY
EXCEPTION 380CH:	WF2 ERR, DRIVE NOT READY
EXCEPTION 380DH:	WF3 ERR, DRIVE NOT READY
EXCEPTION 380EH:	WM0 ERR, DRIVE NOT READY
EXCEPTION 380FH:	WM1 ERR, DRIVE NOT READY

BIOS Exceptions That are not Device Specific

EXCEPTION 3C00H:	DEVICE ERROR, OS FAILURE, BAD COMMAND CODE PASSED TO DRIVER
EXCEPTION 3C10H:	DEVICE ERROR, OS FAILURE, BLOCKS ACCESSED OVERFLOW DEVICE
EXCEPTION 3C20H:	DEVICE ERROR, MINI DRIVER FAILURE, INVALID 8272 COMMAND
EXCEPTION 3C30H:	DEVICE ERROR, OS MINI DRIVER FAILURE, Q IS DAMAGED
EXCEPTION 3C40H:	DEVICE ERROR, OS MINI DRIVER FAILURE, 8089 NOT RESPONDING
EXCEPTION 3C50H:	DEVICE ERROR, ACCESS TO DEVICE W/MISSING CONTROLLER HARDWARE
EXCEPTION 3CF0H:	DEVICE ERROR, OS MINI DRIVER FAILURE, BAD 8272 PROTOCOL
EXCEPTION 3D80H:	KEYBOARD READ ABORTED
EXCEPTION 3DC0H:	DEVICE ERROR, OS HD5440 DRIVER FAIL, BAD CONTROLLER PROTOCOL
EXCEPTION 3DC1H:	DEVICE ERR, OS HD5440 DRIVER FAILURE, Q IS DAMAGED
EXCEPTION 3DC2H:	DEVICE ERR, HD5440 CONTROLLER CONFIGURATION SWITCHES WRONG
EXCEPTION 3DC3H:	DEVICE ERR, UNKNOWN INTERRUPT SOURCE ON MULTIBUS LEVEL @
EXCEPTION 3DC4H:	DEVICE ERR, TRANSFER OVERFLOWS 8086 SEGMENT
EXCEPTION 3DE0H:	ILLEGAL BX_SBIOS ACCESS BY MORE THAN ONE JOB
EXCEPTION 3DF0H:	DEVICE ERR, OS FAILURE, LIST DAMAGED
EXCEPTION 3E01H:	DEVICE ERR, PRINTER TIMEOUT, NO RESPONSE IN 3 SEC
EXCEPTION 3E02H:	DEVICE ERR, PRINTER HARDWARE FAIL, FAULT BEING SIGNALLED
EXCEPTION 3E04H:	DEVICE ERR, PRINTER NOT READY
EXCEPTION 3E10H:	DEVICE ERR, OS PRINTER DRIVER FAIL, PROTOCOL ERROR
EXCEPTION 3F00H:	DEVICE ERR, OS WINC FAIL, DATA STRUCTURE DAMAGED

EXCEPTION 3F01H:	DEVICE ERR, OS WINC DRIVER FAIL, BAD CONTROLLER PROTOCOL
EXCEPTION 3F02H:	DEVICE ERR, UNSUPPORTED WINC CONTROLLER ENCOUNTERED

File Structure Exceptions

EXCEPTION 4000H:	OPEN ATTEMPTED ON CONNECTION WHICH IS ALREADY OPEN
EXCEPTION 4001H:	CONNECTION NOT OPEN
EXCEPTION 4002H:	INCORRECT FILE TYPE
EXCEPTION 4003H:	PARAMETER HAS INVALID SYNTAX
EXCEPTION 4004H:	DEVICE BEING ACCESSED IS NOT READY
EXCEPTION 4006H:	COMMUNICATIONS SYSTEM ERROR
EXCEPTION 4007H:	NETWORK FAILURE—PUBLIC FILES NOT ACCESSIBLE
EXCEPTION 4009H:	OPEN MODE OF CONNECTION PROHIBITS OPERATION
EXCEPTION 4010H:	FILE EXISTS AND HAS CONNECTION ESTABLISHED ON IT
EXCEPTION 4011H:	NONEMPTY DIRECTORY FILE
EXCEPTION 4012H:	MAXIMUM NUMBER OF PERMITTED OBJECTS EXCEEDED
EXCEPTION 4013H:	CONNECTION DOES NOT EXIST
EXCEPTION 4014H:	OPERATION NOT SUPPORTED FOR THE CONSOLE DEVICE
EXCEPTION 4015H:	LOGICAL NAME ALREADY EXISTS
EXCEPTION 4016H:	ILLEGAL DEVICE ID VALUE
EXCEPTION 4017H:	DISMOUNT ATTEMPTED ON SYSTEM DEVICE
EXCEPTION 4018H:	LOGICAL NAME DOES NOT EXIST
EXCEPTION 4019H:	INVALID PATHNAME SYNTAX
EXCEPTION 401AH:	FILE IS WRONG TYPE; DIRECTORY FILE EXPECTED
EXCEPTION 401BH:	FILE DOES NOT EXIST
EXCEPTION 401CH:	FILE EXISTS AND HAS CONNECTION ESTABLISHED ON IT
EXCEPTION 401DH:	EXTERNAL EVENT CAUSED DETACH OF CONNECTION
EXCEPTION 401EH:	FATAL SERIES IV NETWORK PROTOCOL ERROR
EXCEPTION 401FH:	DFS SERIES IV ERROR: E$NOT$LOGGED$ON
EXCEPTION 4020H:	VOLUME WITH SAME VOLUME ROOT DIRECTORY NAME ALREADY MOUNTED
EXCEPTION 8004H:	ILLEGAL PARAMETER VALUE
EXCEPTION DFFEH:	VOLUME DOES NOT EXIST
EXCEPTION DFFFH:	USER OPERATIONS NOT SUPPORTED ON FLIPPIES

Syntax Guide Exceptions

EXCEPTION E000H:	SYNTAX GUIDE LIMIT EXCEEDED: ARGUMENT BUFFER OVERFLOW
EXCEPTION E001H:	SYNTAX GUIDE LIMIT EXCEEDED: PARSE STACK OVERFLOW
EXCEPTION E002H:	SYNTAX GUIDE LIMIT EXCEEDED: REMOVE STACK OVERVIEW
EXCEPTION E003H:	SYNTAX GUIDE LIMIT EXCEEDED: EDIT BUFFER OVERFLOW

EXCEPTION E004H:	SYNTAX TABLES INCOMPATIBLE WITH SYNTAX GUIDE
EXCEPTION E005H:	SYNTAX GUIDE CONSISTENCY CHECK: INTERNAL ERROR

ISIS-IV Exception Codes

EXCEPTION E100H:	INTERNAL ISIS-IV ERROR, BAD MIP CONNECT
EXCEPTION E101H:	INTERNAL ISIS-IV ERROR, UNKNOWN MONITOR REQUEST TYPE
EXCEPTION E103H:	INTERNAL ISIS-IV ERROR, UNKNOWN MESSAGE DESTINATION
EXCEPTION E107H:	INTERNAL ISIS-IV ERROR, NO RECEIVE BUFFERS
EXCEPTION E109H:	INTERNAL ISIS-IV ERROR, CLOSED DESTINATION PORT
EXCEPTION E10BH:	IEU BOARD NOT RESPONDING (DAMAGED SOFTWARE POSSIBLE)
EXCEPTION E10DH:	INTERNAL ISIS-IV ERROR, BAD MESSAGE ADDRESS
EXCEPTION E111H:	IEU HARDWARE NOT RESPONDING
EXCEPTION E112H:	INTERNAL ISIS-IV ERROR, UNKNOWN MESSAGE TYPE
EXCEPTION E113H:	INTERNAL ISIS-IV ERROR, UNKNOWN ISIS REQUEST TYPE
EXCEPTION E114H:	INTERNAL ISIS-IV ERROR, BAD LNAME
EXCEPTION E115H:	CANNOT LOAD ISIS. LM
EXCEPTION E116H:	READ FROM KEYBOARD IN BACKGROUND OR IMPORT WAS ATTEMPTED
EXCEPTION E117H:	ERROR OCCURED WHILE WRITING TO :CO: OR WHILE :CO: WAS CLOSED
EXCEPTION E118H:	INTERNAL ISIS-IV ERROR, BAD CONSOLE AFTN
EXCEPTION E119H:	UNSUPPORTED MONITOR FUNCTION WAS CALLED
EXCEPTION E11AH:	BAD PARAMETER TO MONITOR KIC ROUTINE
EXCEPTION E11BH:	ISIS-IV ERROR: BAD REMOTE INTERFACE COMMAND
EXCEPTION E11CH:	ISIS-IV ERROR: REMOTE INTERFACE PROTOCOL VIOLATION
EXCEPTION E11DH:	ISIS-IV ERROR: IEU MEMORY NOT AVAILABLE TO ISIS-IV
EXCEPTION E11EH:	ISIS-IV ERROR: I/O ERROR ON CONSOLE FILE
EXCEPTION E200H:	INSUFFICIENT MEMORY AVAILABLE FOR VIEW

File Structure CUSP Exceptions

EXCEPTION FOOOH:	FATAL FILE SYSTEM CONSISTENCY ERROR
EXCEPTION FFE9H:	FATAL FILE SYSTEM CONSISTENCY ERROR
EXCEPTION FFEBH:	INCORRECT PASSWORD
EXCEPTION FFECH:	USER NAME NOT KNOWN
EXCEPTION FFECH:	LEGAL USER ID VALUE
EXCEPTION FFEEH:	OPERATION LIMITED TO SUPER USER/MASTER SUPER USER
EXCEPTION FFEFH:	USER OR HOME DIRECTORY DEFINITION FILE DOES NOT EXIST
EXCEPTION FFF2H:	USER ID ALREADY EXISTS
EXCEPTION FFF3H:	USER NAME ALREADY EXISTS OR USER SPECIFICATION MISMATCH

EXCEPTION FFF4H: SYSTEM FILE INACCESSIBLE
EXCEPTION FFFAH: MAXIMUM FILE LENGTH EXCEEDED

UDI-Series IV Exception Codes

Exceptions Returned From System Calls

E$OK (0000H)

E$ABS (2303H)

Program contains an absolute record.

E$ACTIVE

Open connections to the volume existed and had to be forcibly detached.

E$ADDRESS

The overlay loaded contained addresses in the operating system area. The load was not completed.

E$BAD$FILE (2301H)

The file containing the overlay is not a valid object file.

E$COMM$ERROR

An error occurred in the communication system.

E$CONNECTION$EXIST

Connections to the volume existed and were detached.

E$CONSOLE

An attempt was made to interface a connection with the console.

E$CONTEXT (0101H)

The routine was called in an illegal context. More specifically, this includes an attempt to:

- Attach, create, or delete a file to which the console was assigned
- Attach, create, or delete the user file containing overlays
- Attach, create, delete or rename a file at the Network Manager when the user is no longer logged on. (Applies only to NDS-II workstations.)

E$CROSSFS (0102H)

The operation attempted an illegal cross volume rename.

E$DEVICE$IO$ERROR

An I/O error occurred in the machinery.

E$DEVICE$NOT$READY

The device is not ready for usage.

E$DIR$NOT$EMPTY

The target file is not an empty directory file.

E$EXIST (0103H)

The specified token or connection did not exist, or the specified overlay did not exist.

E$FACCESS (0026H)

A deletion, rename, or destructive creation of a write-protect or format file was attempted; or the mode of open did not agree with the file attributes or device characteristics.

E$FEXIST (0020H)

The specified file exists when it is not expected to exist.

E$FNEXIST (0021H)

The specified file does not exist when it is expected to exist. The deletion, rename, or attachment of a workfile will also cause the exception.

E$FTYPE

A non-terminal path component is not a directory file.

E$ILL$RECORD (2302H)

Illegal OMF record detected by the loader.

E$ILL$VOLUME

An illegal volume-name was specified.

E$LIMIT

The calling job has exceeded the allowable number of attaches.

E$MARKED$DELETED

The file has been marked for deletion and cannot be attached.

E$MEM (0002H)

Insufficient memory for requested operation.

E$MOUNTED

The device is already mounted.

E$NODE$NOT$READY

The node is not responding to the request.

E$NOPEN (0104H)

The operation attempted to close, read, write, or seek a connection that was not opened.

E$OPEN (0105H)

The operation attempted to open a connection that was already opened.

E$OPEN$MODE

The opened mode does not allow reading to occur.

E$OREAD (0106H)

A write operation was attempted on a connection opened for read.

E$OWRITE (0107H)

A read operation was attempted on a connection opened for write.

E$PARAM (0108H)

An argument had an illegal value. This is usually the result of a bounds check (e.g., $0 \leq$connection token≤ 12).

E$PTR (0109H)

A pointer argument was illegal. If this was the excep$p argument, the operating system aborts the job and prints an error message to the cold-start console.

E$SHADOWED

The (local) volume has been mounted, but shadows the corresponding public volume.

E$SHARE (0028H)

An attempt was made to delete, rename, open, destructively create, or attach to a file on which a connection was already established. This may mean that the file is currently open by another user.

E$SIX (010AH)

An attempt was made to open a seventh connection.

E$SPACE (0029H)

The operation attempted to add a directory entry to a full directory.

E$STRING$BUF (0081H)

The string is over 45 characters long (DQ$CHANGE$EXTENSION) or the argument is over 80 characters long (DQ$GET$ARGUMENT).

E$SUPPORT (23H)

One of the following operations was attempted:

- The deletion or renaming of a physical or logical device
- The seeking of a physical device or the console

- A DQ$SPECIAL with a connection that was not established on :CI:
- A DQ$SPECIAL with type 1 or type 3 when :CI: has been assigned to a disk file.

E$SYNTAX	(010CH)

An illegal ISIS pathname was specified. This includes device-name parts not supported by Series IV, or an illegal overlay name.

E$SYSTEM$DEVICE

An attempt was made to dismount a system device.

Hardware-Detected Conditions

E$ZERO$DIVIDE	(8000H)

A divide by zero was attempted.

E$OVERFLOW	(8001H)

An overflow occurred.

E$8087	(8007H)

An 8087 error occurred.

APPENDIX D OBJECT MODULE RELOCATION AND LINKAGE

Figure D-1 presents the valid possibilities for combining object modules created by resident and cross-product translators or by relocation-and-linkage packages. ("Cross-product" here means software packages that execute on an 8080/8085-based system but create code to run on an 8086/8088 based system.)

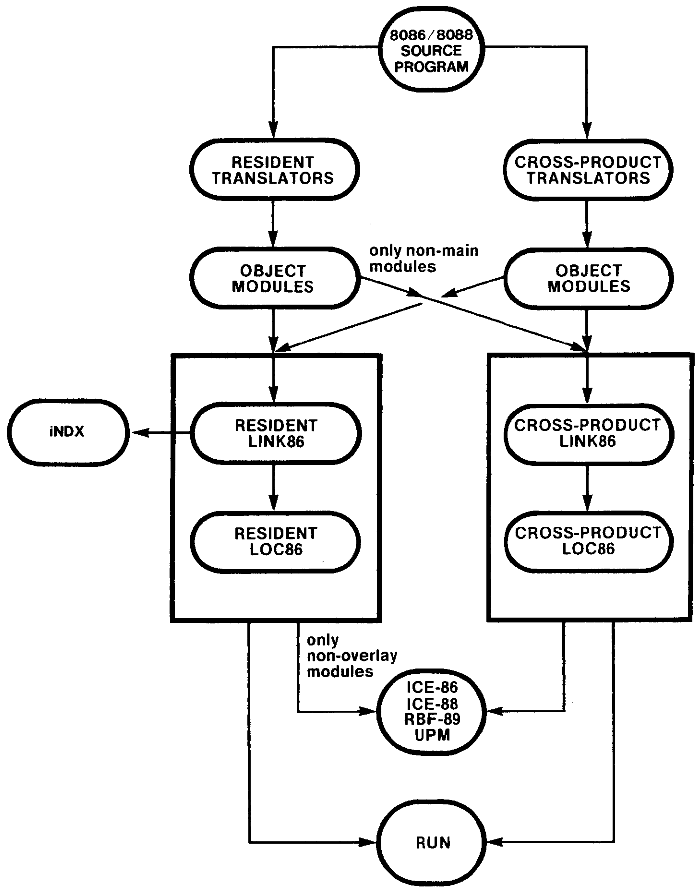


Figure D-1. Use of Relocation and Linkage Packages 121753-10

As long as LINK and LOCATE are used in sequence, object modules developed using cross-product translators can be combined with new object modules developed using resident translators. The permitted and prohibited possibilities are as follows:

Permitted

1. The resident R&L86 will successfully process any object module produced by any Intel cross-product or resident product.
2. The cross-product R&L package will successfully process any non-main object module produced by resident translators.
3. Current absolute loaders (e.g., those named in figure D-1) will successfully process non-overlay modules produced by resident products as well as the output of cross-products.
4. Position-Independent Code (PIC) or Load-Time Locatable (LTL) modules do not need LOC86 and can be executed directly after being processed by LINK86.

Prohibited

1. The cross-product R&L package cannot process the following kinds of modules:
   a. Main object modules from resident translators.
   b. Modules produced by resident R&L86.
2. Current absolute loaders cannot process overlay modules produced by resident R&L86.

See the *iAPX 86, 88 Family Utilities User's Guide*, 121616, for complete details on Intel's R&L packages.

NOTE

The Series IV will *not* load a module with *any* absolute records, i.e., a module that has been "located," variables declared at an absolute location (@), or a procedure declared "interrupt."

APPENDIX E BOOT DEVICE-CONFIGURATION SWITCH ASSIGNMENTS

Table E-1. Boot Device Assignment

Switch Label					Description
1	2	34	567	8	
0[1]	x	00	111	1	Boot Diagnostic OS from comm
0	x	—	—	1	Boot OS from comm (see note 3)
0	x	00	000	0	Jump to Mon88, no power-up test (see note 4)
0	x	00	001	0	Boot from 5 1/4″ floppy disk - dr 0 (see note 5)
0	x	01[2]	001	0	Boot from 5 1/4″ floppy disk - dr 1 (see note 5)
0	x	00	010	0	Boot from Hard disk - dr 0 (see note 5)
0	x	01	010	0	Boot from Hard disk - dr 1 (see note 5)
0	x	00	011	0	Boot from Priam (35 MB Winchester)
0	x	00	101	0	Boot from Integrated 5 1/4″ Winchester
0	x	00	110	0	Reserved
0	x	00	111	0	Reserved

NOTES

1. 0 = Off (down)
2. 1 = On (up)
3. If switch 8 is set to 1, the firmware will attempt to boot from the network. If the network boot fails, an attempt is made to boot from the local system device as specified by switches 3 to 7.
4. If switches 5, 6, 7, and 8 are set to zero, the bootstrap firmware will transfer control to Mon88. The power on confidence test will not be run.
5. If the local boot fails, control is transferred to Mon88.

APPENDIX F
ASCII CODES

Table F-1. ASCII Code List

Decimal	Octal	Hexadecimal	Character
0	000	00	NUL
1	001	01	SOH
2	002	02	STX
3	003	03	ETX
4	004	04	EOT
5	005	05	ENQ
6	006	06	ACK
7	007	07	BEL
8	010	08	BS
9	011	09	HT
10	012	0A	LF
11	013	0B	VT
12	014	0C	FF
13	015	0D	CR
14	016	0E	SO
15	017	0F	SI
16	020	10	DLE
17	021	11	DC1
18	022	12	DC2
19	023	13	DC3
20	024	14	DC4
21	025	15	NAK
22	026	16	SYN
23	027	17	ETB
24	030	18	CAN
25	031	19	EM
26	032	1A	SUB
27	033	1B	ESC
28	034	1C	FS
29	035	1D	GS
30	036	1E	RS
31	037	1F	US
32	040	20	SP
33	041	21	!
34	042	22	"
35	043	23	#
36	044	24	$
37	045	25	%
38	046	26	&
39	047	27	'
40	050	28	(
41	051	29	)
42	052	2A	*
43	053	2B	+
44	054	2C	,
45	055	2D	—
46	056	2E	.
47	057	2F	/
48	060	30	0
49	061	31	1
50	062	32	2
51	063	33	3
52	064	34	4
53	065	35	5
54	066	36	6
55	067	37	7
56	070	38	8
57	071	39	9
58	072	3A	:
59	073	3B	;
60	074	3C	<

Table F-1. ASCII Code List (Cont'd.)

Decimal	Octal	Hexadecimal	Character
61	075	3D	=
62	076	3E	>
63	077	3F	?
64	100	40	@
65	101	41	A
66	102	42	B
67	103	43	C
68	104	44	D
69	105	45	E
70	106	46	F
71	107	47	G
72	110	48	H
73	111	49	I
74	112	4A	J
75	113	4B	K
76	114	4C	L
77	115	4D	M
78	116	4E	N
79	117	4F	O
80	120	50	P
81	121	51	Q
82	122	52	R
83	123	53	S
84	124	54	T
85	125	55	U
86	126	56	V
87	127	57	W
88	130	58	X
89	131	59	Y
90	132	5A	Z
91	133	5B	[
92	134	5C	\
93	135	5D	]
94	136	5E	^
95	137	5F	_
96	140	60	`
97	141	61	a
98	142	62	b
99	143	63	c
100	144	64	d
101	145	65	e
102	146	66	f
103	147	67	g
104	150	68	h
105	151	69	i
106	152	6A	j
107	153	6B	k
108	154	6C	l
109	155	6D	m
110	156	6E	n
111	157	6F	o
112	160	70	p
113	161	71	q
114	162	72	r
115	163	73	s
116	164	74	t
117	165	75	u
118	166	76	v
119	167	77	w
120	170	78	x
121	171	79	y
122	172	7A	z
123	173	7B	{
124	174	7C	|
125	175	7D	}
126	176	7E	~
127	177	7F	DEL

Table F-2. ASCII Code Definition

Abbreviation	Meaning	Decimal Code
NUL	NULL Character	0
SOH	Start of Heading	1
STX	Start of Text	2
ETX	End of Text	3
EOT	End of Transmission	4
ENQ	Enquiry	5
ACK	Acknowledge	6
BEL	Bell	7
BS	Backspace	8
HT	Horizontal Tabulation	9
LF	Line Feed	10
VT	Vertical Tabulation	11
FF	Form Feed	12
CR	Carriage Return	13
SO	Shift Out	14
SI	Shift In	15
DLE	Data Link Escape	16
DC1	Device Control 1	17
DC2	Device Control 2	18
DC3	Device Control 3	19
DC4	Device Control 4	20
NAK	Negative Acknowledge	21
SYN	Synchronous Idle	22
ETB	End of Transmission Block	23
CAN	Cancel	24
EM	End of Medium	25
SUB	Substitute	26
ESC	Escape	27
FS	File Separator	28
GS	Group Separator	29
RS	Record Separator	30
US	Unit Separator	31
SP	Space	32
DEL	Delete	127

INDEX

REQUEST FOR READER'S COMMENTS

Intel's Technical Publications Departments attempt to provide publications that meet the needs of all Intel product users. This form lets you participate directly in the publication process. Your comments will help us correct and improve our publications. Please take a few minutes to respond.

Please restrict your comments to the usability, accuracy, readability, organization, and completeness of this publication. If you have any comments on the product that this publication describes, please contact your Intel representative. If you wish to order publications, contact the Intel Literature Department (see page ii of this manual).

1. Please describe any errors you found in this publication (include page number).

____
____
____
____
____
____

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

____
____
____
____
____

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

____
____
____
____
____
____

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

____
____
____
____

5. Please rate this publication on a scale of 1 to 5 (5 being the best rating). ____________

NAME ______________________________ DATE ____________
TITLE ____
COMPANY NAME/DEPARTMENT ____________________________
ADDRESS ____
CITY ________________ STATE ________________ ZIP CODE ____________
(COUNTRY)

Please check here if you require a written reply. ☐

WE'D LIKE YOUR COMMENTS ...

This document is one of a series describing Intel products. Your comments on the back of this form will help us produce better manuals. Each reply will be carefully reviewed by the responsible person. All comments and suggestions become the property of Intel Corporation.

NO POSTAGE
NECESSARY
IF MAILED
IN U.S.A.

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 1040 SANTA CLARA, CA

POSTAGE WILL BE PAID BY ADDRESSEE

Intel Corporation
Attn: Technical Publications M/S 6-2000
3065 Bowers Avenue
Santa Clara, CA 95051

intel®

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

Printed in U.S.A.