

HP 3000 Series II Computer System



MPE Commands

Reference Manual



5303 STEVENS CREEK BLVD., SANTA CLARA, CALIFORNIA, 95050

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

NOTICE

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

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

This document contains proprietary information which is protected by copyright. All rights are reserved. No part of this document may be photocopied, reproduced or translated to another program language without the prior written consent of Hewlett-Packard Company.

LIST OF EFFECTIVE PAGES

The List of Effective Pages gives the most recent date on which the technical material on any given page was altered. If a page is simply re-arranged due to a technical change on a previous page, it is not listed as a changed page. Within the manual, changes are marked with a vertical bar in the margin.

Pages	Effective Date
ii to xiv	June 1976
1-1 to 1-5	June 1976
2-1 to 2-157	June 1976
3-1 to 3-31	June 1976
4-1 to 4-21	June 1976
5-1 to 5-8	June 1976
6-1 to 6-85	June 1976
7-1 to 7-19	June 1976
8-1 to 8-15	June 1976
9-1 to 9-12	June 1976
10-1 to 10-18	June 1976
A-1	June 1976
B-1 to B-9	June 1976
I-1 to I-7	June 1976

PRINTING HISTORY

New editions incorporate all update material since the previous edition. Update packages, which are issued between editions, contain additional and replacement pages to be merged into the manual by the customer. The date on the title page and back cover changes only when a new edition is published. If minor corrections and updates are incorporated, the manual is reprinted but neither the date on the title page and back cover nor the edition change.

First Edition June 1976

PREFACE

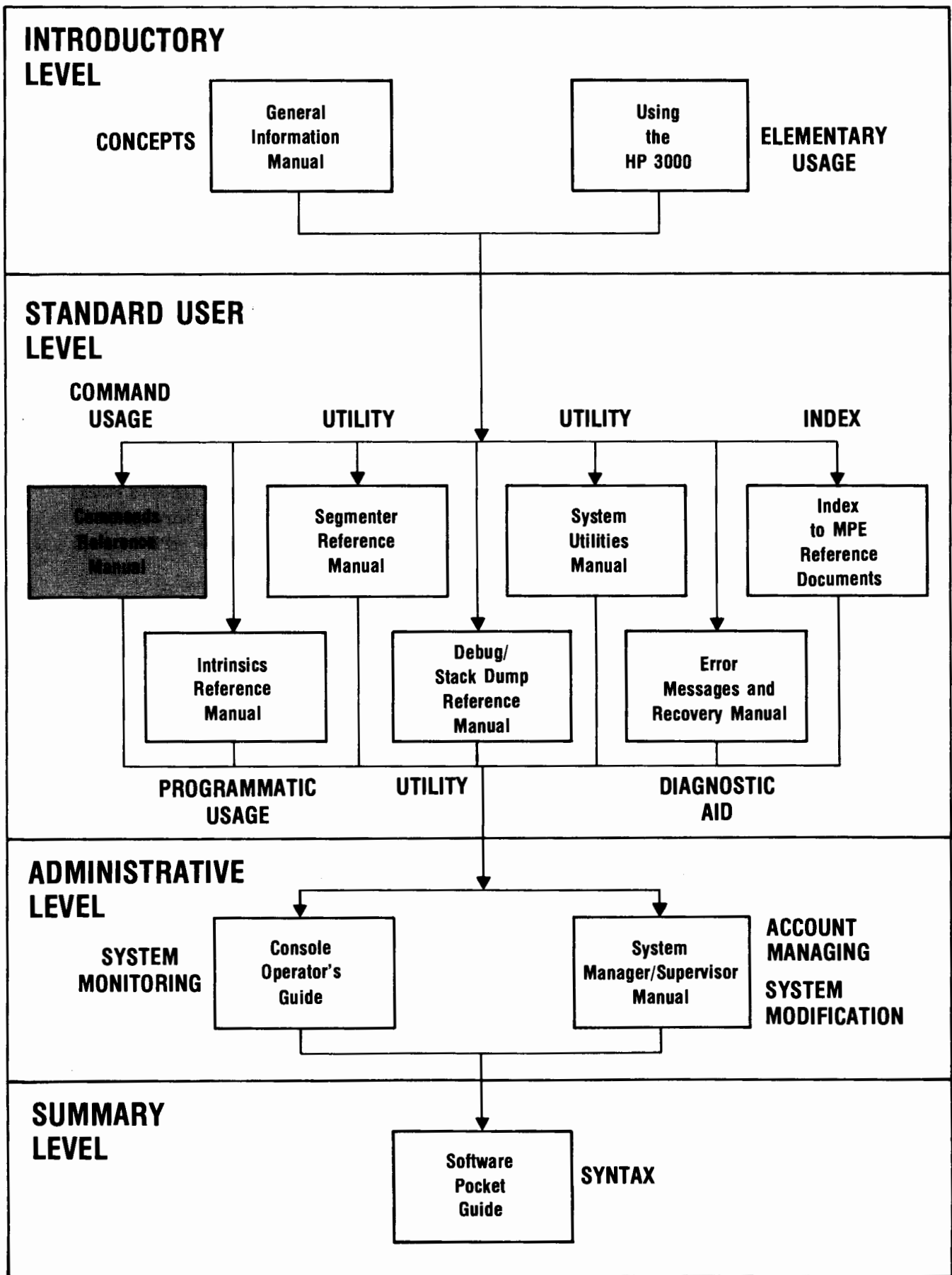
This manual is one of a set of twelve manuals that document the Multiprogramming Executive Operating System (MPE-II) for the HP 3000 Series II Computer System. It explains how to issue MPE commands for controlling the processing of programs. Specifically, this manual shows you how to: initiate interactive sessions and batch jobs for processing programs; manage files used by those programs; compile, prepare, and run the programs; determine the status of sessions, jobs, files, devices, and other elements; and perform various utility functions. As an aid in debugging, this manual also explains the command-related diagnostic messages that are issued by the operating system. The Guide For Readers, presented on page xii, shows you what sections of the manual to read to perform specific tasks.

This manual is a reference book rather than a tutorial text for new programmers. To use it effectively, you should understand the fundamental techniques of programming. You should also obtain an overview of the relations between the main hardware and software features of the HP 3000 Series II Computer System by reading the *General Information Manual* (part number 30000-90008). No other prerequisites are necessary.

The Manual Plan following this preface shows the relationship of this manual (shaded block) to others in the MPE manual set. Further information about the Manual Plan appears in *Index to MPE Reference Documents* (part number 30000-90045). The entire MPE manual set contains:

Manual Title	Part No.
Using the HP 3000: A Guide for the Terminal User	03000-90121
HP 3000 Software Pocket Guide	03000-90126
General Information Manual	30000-90008
MPE Commands Reference Manual	30000-90009
MPE Intrinsic Reference Manual	30000-90010
MPE Segmenter Reference Manual	30000-90011
MPE Debug/Stack Dump Reference Manual	30000-90012
Console Operator's Guide	30000-90013
System Manager/System Supervisor Manual	30000-90014
System Error Messages and Recovery Manual	30000-90015
MPE System Utilities Manual	30000-90044
Index to MPE Reference Documents	30000-90045
HP 3000CX to HP 3000 Series II Program Conversion Guide	30000-90046

MANUAL PLAN



CONVENTIONS USED IN THIS MANUAL

NOTATION

DESCRIPTION

[]	An element inside brackets is <i>optional</i> . Several elements stacked inside a pair of brackets means the user may select any one or none of these elements. Example: $\begin{bmatrix} A \\ B \end{bmatrix}$ user may select A or B or neither
{ }	When several elements are stacked within braces the user must select one of these elements. Example: $\begin{Bmatrix} A \\ B \\ C \end{Bmatrix}$ user must select A or B or C.
italics	Lowercase italics denote a parameter which must be replaced by a user-supplied variable. Example: CALL <i>name</i>
underlining	Dialogue: Where it is necessary to distinguish user input from computer output, the input is underlined. Example: NEW NAME? <u>ALPHA1</u>
superscript C	Control characters are indicated by a superscript C Example: Y ^C
<i>return</i>	<i>return</i> in italics indicates a carriage return
<i>linefeed</i>	<i>linefeed</i> in italics indicates a linefeed
...	A horizontal ellipsis indicates that a previous bracketed element may be repeated, or that elements have been omitted.

CONTENTS

Section I	Page	Section II	Page
INTRODUCTION TO COMMANDS			
How To Use This Manual	1-1	:SAVE	2-119
How To Enter Commands	1-2	:SECURE	2-121
Command Elements	1-2	:SEGMENTER	2-123
Positional Parameters	1-3	:SETDUMP	2-125
Keyword Parameters	1-4	:SETMSG	2-127
Continuation Characters	1-5	:SHOWDEV	2-129
Command Errors	1-5	:SHOWIN	2-131
Executing Commands Programmatically	1-5	:SHOWJOB	2-135
		:SHOWOUT	2-137
Section II	Page	:SHOWTIME	2-141
COMMAND LANGUAGE		:SPEED	2-143
:ABORT	2-5	:SPL	2-145
:ALTSEC	2-7	:SPLGO	2-147
:BASIC	2-9	:SPLPREP	2-149
:BASICGO	2-11	:STORE	2-151
:BASICOMP	2-13	:STREAM	2-153
:BASICPREP	2-15	:TELL	2-155
:BUILD	2-17	:TELLOP	2-157
:BYE	2-21		
:COBOL	2-23	Section III	Page
:COBOLGO	2-25	RUNNING SESSIONS	
:COBOLPREP	2-27	Logging-On	3-4
:COMMENT	2-29	Information Always Required	3-5
:CONTINUE	2-31	Information Sometimes Required	3-7
:DATA	2-33	Group Name	3-7
:DEBUG	2-35	Passwords and Their Security	3-8
:EDITOR	2-37	Terminal Types	3-9
:EOD	2-39	Optional Information	3-10
:EOF	2-41	Session Name	3-10
:EOJ	2-43	Time Limit	3-10
:FILE	2-45	Scheduling	3-10
:FORTGO	2-55	Errors During Log-On	3-13
:FORTPREP	2-57	Logging-Off	3-14
:FORTRAN	2-59	Interrupting Command Execution	3-15
:FREERIN	2-61	Interrupting Non-Program Commands	3-16
:GETRIN	2-63	Interrupting Program Commands	3-16
:HELLO	2-65	Aborting a Program	3-18
:JOB	2-69	Reading Data from Standard Input Device	3-19
:LISTF	2-75	Effects of Spooling	3-20
:PREP	2-79	Premature Session Termination or Suspension	3-22
:PREPRUN	2-83	Error Handling	3-23
:PTAPE	2-87	Using a Terminal	3-24
:PURGE	2-89	Log-On Characteristics	3-24
:RELEASE	2-91	Echo Facility and Transmission Mode	3-25
:RENAME	2-93	Special Keyboard Functions	3-25
:REPORT	2-95	Special Terminal Types for	
:RESET	2-97	HP 2640 and HP 2644 Terminals	3-29
:RESETDUMP	2-99	Character Mode (;TERM=10)	3-29
:RESTORE	2-101	Block Mode (;TERM=11)	3-29
:RESUME	2-105		
:RJE	2-107	Section IV	Page
:RPG	2-109	RUNNING JOBS	
:RPGGO	2-111	Initiating a Job	4-4
:RPGPREP	2-113	Information Always Required	4-4
:RUN	2-115		

CONTENTS (continued)

Information Sometimes Required	4-5
Optional Information	4-6
Job Name	4-6
Central Processor Time Limit	4-7
Execution Priority	4-7
Input Priority	4-7
Re-Starting Interrupted Jobs	4-8
Controlling Job Output	4-8
Errors During Job Entry	4-10
Terminating a Job	4-10
Reading Programs and Data From	
Standard Input Device	4-11
Introducing Jobs Within a Session	4-12
Inserting Comments in a Job	4-16
Effects of Spooling	4-17
Premature Job Termination or Suspension	4-19
Error Handling	4-21

Section V	Page
DETERMINING SESSION/JOB STATUS AND RESOURCE USAGE	
How MPE Handles Sessions and Jobs	5-1
How MPE Handles Resource Accounting	5-6
Resource-Use Limits	5-6
Resource Accounting	5-7

Section VI	Page
MANAGING FILES	
Disc Files and Devicefiles	6-3
How MPE Handles File Processing	6-3
Opening a File	6-4
Transferring Data to and from a File	6-8
Disc Files	6-8
Devicefiles	6-8
Closing a File	6-11
File Domains	6-12
Specifying File Characteristics	6-12
Summary of General Rules	6-17
Specifying File Designators	6-18
User-Defined Files	6-18
System-Defined Files	6-22
Input/Output Sets	6-23
Searching File Domains	6-24
Devices	6-25
Logical Records and Blocks	6-27
Fixed-Length Records	6-27
Variable-Length Records	6-29
Undefined-Length Records	6-30
Device Constraints on Records and Blocks ..	6-31
ASCII vs. Binary Code	6-32
Improving Input/Output Efficiency	6-33
Conserving Disc Space	6-33
Reading Tapes with Unknown Contents	6-33
Submitting Jobs and Data from Magnetic	
Tape Files	6-34
Padding Records	6-34
Input/Output Buffers	6-34

Section VI	Page
Multi-Record Mode	6-36
Restricting File Access (By Access-Mode)	6-36
Carriage-Control Characters	6-38
File Codes	6-39
NOWAIT Input/Output	6-40
Disc File Considerations	6-40
Determining File Space Required	6-41
Allocating File Space	6-42
Simultaneous Access of Files	6-44
Closing Disposition	6-49
:FILE/FOPEN Parameter Relations	6-49
Device-Dependent Characteristics	6-51
Headers and Trailers	6-51
Special Forms	6-53
Implied :FILE Commands	6-53
Cancelling :FILE Commands	6-54
Copying Files	6-54
Managing Disc Files	6-54
Creating a New File	6-55
Listing File Descriptions	6-55
Renaming a File	6-60
Adding, Removing, or Changing Lockwords	6-60
Transferring Files	6-61
Inter-Group Transfers	6-61
Inter-Account Transfers	6-61
Inter-System Transfers	6-62
Deleting a File	6-62
Saving a File	6-63
Dumping Files Off-Line	6-64
Magnetic Tape Format	6-66
Listing Output	6-68
Examples	6-70
Retrieving Dumped Files	6-71
Listing Output	6-71
Example	6-74
Specifying Disc File Security	6-74
Account-Level Security	6-76
Group-Level Security	6-77
File-Level Security	6-78
Changing Security Provisions	6-79
Suspending and Restoring Security Provisions ..	6-80
Reading Data from Outside Standard	
Input Stream	6-81
End-of-File Indicators	6-83
Command/File-Type Summary	6-85

Section VII	Page
RUNNING SUBSYSTEMS AND USER PROGRAMS	
Compiling, Preparing, and Executing Programs	7-1
Compilation	7-1
Program Preparation	7-3
Allocation/Execution	7-3
Compilation Only	7-4
Compilation/Preparation	7-6
Compilation/Preparation/Execution	7-8

CONTENTS (continued)

Preparation Only	7-9
Preparation/Execution	7-11
Execution Only	7-12
Using the Basic Interpreter	7-14
Using the Basic Compiler	7-15
Using Other Subsystems	7-15
Implicit :FILE Commands for Subsystems	7-16

Section VIII

DETERMINING DEVICE AND DEVICEFILE STATUS

How MPE Handles Devices	8-1
How MPE Handles Devicefiles	8-4
Input Devicefiles	8-4
Session/Job Input Devicefiles	8-4
Data Devicefiles	8-5
Operator-Assigned Devicefiles	8-5
Accessing Devicefiles	8-5
Input Spooling	8-6
Obtaining Information About Input Devicefiles	8-8
Output Devicefiles	8-10
Session/Job Listing Devicefiles	8-10
Other Output Devicefiles	8-10
Accessing Devicefiles	8-11
Output Spooling	8-12
Obtaining Information About Output Devicefiles	8-13

Section IX

Page

REQUESTING UTILITY OPERATIONS

Communication Between Users	9-1
User-to-User Communication	9-1
User-to-Operator Communication	9-4
Operator-to-User Communication	9-4
Changing Terminal Speed	9-5
Using Paper Tapes at the Terminal	9-6
Managing Resources through Resource Identification Numbers (RINS)	9-10
Acquiring Global RINS	9-10
Freeing Global RINS	9-11
Displaying Current Date and Time	9-12

Section X

Page

MESSAGES

Command Interpreter Error Messages	10-2
Command Interpreter Warning Messages	10-15
System Messages	10-17
Operator Messages	10-18
User Messages	10-18

Appendix A

TERMINALS SUPPORTED BY MPE

Appendix B

DETAILS OF PROGRAM EXECUTION

ILLUSTRATIONS

Title	Page	Title	Page
A Typical Session	3-2	Program Management	7-1
Echo Facility/Transmission Mode	3-26	Listing of Prepared Program	7-10
A Typical Job	4-3	Listing of Loaded Program	7-13
File Analogy	6-2	Input/Output Spooling	8-7
File Specification Hierarchy	6-6	User/Operator Intercommunication	9-2
Sequential Writing to New Disc File	6-9	X-OFF/X-ON Hand-Shaking Arrangement	9-7
MPE File Structure	6-10	:PTAPE Command Operation	9-8
Formal vs. Actual File Designators	6-16	Code-Sharing and Data Privacy	B-2
Disc Space Used by Fixed-Length Records	6-28	Code Segment and Associated	
Disc Space Used by Variable-Length Records	6-30	Registers	B-4
Checks for File Dump Eligibility	6-65	Data (Stack) Segment and Associated	
:STORE Tape Format	6-66	Registers	B-5
List Output Format of :STORE Command	6-69	Stack Operation	B-9
List Output of :RESTORE with SHOW			
and KEEP	6-72		



TABLES

Title	Page	Title	Page
Functional List of Commands	2-2	Disc File Label Contents	6-58
Reference Notes for Command Definitions	2-4	:STORE Tape Format	6-67
Preparing Input/Output Devices	3-6	:STORE Command Error Messages	6-70
Session/Job Priority Classes	3-11	:RESTORE Command Error Messages	6-73
Non-Program Commands	3-15	Access Modes for Intrinsic	6-79
Program Commands	3-15	File-Terminating Entries	6-84
Special Functions and Keys	3-27	Commands vs. File Types	6-85
System-Defined File Designators	6-22	Compilation/Preparation Commands	7-6
Input Set	6-23	Compilation/Preparation/Execution	
Output Set	6-23	Commands	7-8
File Domains Permitted	6-24	Prepared Program Listing Key	7-11
Device Configurations	6-25	Loaded Program Listing Key	7-14
Standard Default Record Sizes	6-32	Subsystem Formal File Designators	7-17
File Access Types	6-37	Preparation/Execution and Store/Restore	
Effects of Access Modes	6-38	Command Formal File Designators	7-19
File-Sharing Restriction Options	6-45	Command Interpreter Error Messages	10-3
Actions Resulting from Multi-Access		Command Interpreter Warning Messages	10-16
of Files	6-46	System Messages	10-17
:FILE vs. FOPEN Parameters	6-50	Data Areas in Stack Segment	B-6
Device-Dependent Restrictions	6-52		

GUIDE FOR READERS

Task	Section
Initiate and run interactive sessions.	III
Initiate and run batch jobs.	IV
Determine session/job status.	V
Determine resource usage.	V
Create, save, and retrieve files.	VI
Specify file security.	VI
Read data outside standard input stream.	VI
Compile, prepare, and run programs.	VII
Use the BASIC Interpreter and Compiler.	VII
Determine device and devicefile status.	VIII
Communicate with other users and the Console Operator.	IX
Change terminal speed on-line.	IX
Read paper tapes without X-OFF control.	IX
Acquire resource identification numbers.	IX
Diagnose errors.	X

FUNCTIONAL LIST OF COMMANDS

FUNCTION	COMMAND
Running Sessions (Section III)	<ul style="list-style-type: none"> :HELLO :BYE :ABORT :RESUME :EOD :EOF:
Running Jobs (Section IV)	<ul style="list-style-type: none"> :JOB :EOJ :STREAM :COMMENT :CONTINUE
Determining Session/Job Status and Resource Usage (Section V)	<ul style="list-style-type: none"> :SHOWJOB :REPORT
Managing Files (Section VI)	<ul style="list-style-type: none"> :FILE . :RESET :BUILD :LISTF :RENAME :PURGE :SAVE :STORE :RESTORE :ALTSEC :RELEASE :SECURE :DATA
Running Subsystems and User Programs (Section VII; commands marked with asterisk (*) are discussed in pertinent subsystem manuals.)	<ul style="list-style-type: none"> :BASIC :BASICOMP :BASICPREP :BASICGO :COBOL :COBOLPREP :COBOLGO :FORTRAN :FORTPREP :FORTGO :RPG :RPGPREP :RPGGO :SPL :SPLPREP :SPLGO :PREP :PREPRUN :RUN :DEBUG* :EDITOR* :RJE* :SEGMENTER* :SETDUMP* :RESETDUMP*

FUNCTIONAL LIST OF COMMANDS (Continued)

FUNCTION	COMMAND
Determining Device and Devicefile Status (Section VIII)	{ :SHOWDEV :SHOWIN :SHOWOUT
Requesting Utility Operations (Section IX)	{ :TELL :TELLOP :SETMSG :SPEED :PTAPE :GETRIN :FREERIN :SHOWTIME

MPE commands allow you to initiate, control, and terminate the processing of programs and to request various other system operations. You generally use them for functions external to the source-language programs that you write, although many of these functions may be necessary to support those programs. For example, you use commands for:

- Logging-on to the system by initiating an interactive session (:HELLO command) or batch job (:JOB command).
- Displaying status information about sessions and jobs in the system (:SHOWJOB).
- Creating, saving, and deleting files (:BUILD, :SAVE, and :PURGE, respectively); specifying and listing their characteristics (:FILE and :LISTF); dumping them offline and subsequently restoring them to the system (:STORE and :RESTORE); and specifying security provisions for them (:ALTSEC, :RELEASE, and :SECURE).
- Compiling programs (:FORTRAN, :COBOL, :RPG, :SPL), preparing those programs (:PREP), and executing them (:RUN).
- Determining the status of devices (:SHOWDEV).
- Determining the status of *devicefiles*, which are disc files originating on or destined for non-sharable devices (:SHOWIN and :SHOWOUT).
- Communicating with other users (:TELL) and with the Console Operator (:TELLOP).

Many other commands and functions are available as well; complete specifications for all of the commands appear in Section II.

HOW TO USE THIS MANUAL

This manual employs two approaches in describing MPE commands and operations: *reference specifications* and *text discussion*.

The *reference specifications*, appearing in Section II, primarily cover the rules for entering each command. Specifically, they show the command syntax and format; define the parameters and discuss constraints upon them and default values for them; provide an overview of the operation requested by the command; and present examples illustrating proper command entries. Parameter definitions common to more than one command are repeated for each applicable command, to reduce the amount of cross-referencing you must do when looking up a definition.

NOTE

If you are already familiar with this or a previous version of MPE, you can use these specifications immediately to look up rules covering the commands you wish to use. **BUT, IF YOU HAVE NO PRIOR EXPERIENCE WITH MPE, YOU SHOULD READ THE TEXT DISCUSSION (SECTIONS III THROUGH X) BEFORE ATTEMPTING TO USE THE REFERENCE SPECIFICATIONS IN ACTUAL APPLICATIONS.**

The *text discussion*, appearing in Sections III through X, explains the functions available through MPE, when and why you would need them, and how to request them through sequences of commands. It also deals with possible errors that you might encounter and how to avoid them. If you are a new user, you will find this discussion helpful in giving you the MPE background you will require at the command level. If you are already familiar with MPE, but need more operational details than those presented in the reference specifications, you will also find these sections useful.

The reference specifications contain cross-references to all associated material in the text discussion, so that you can readily find all information pertinent to each command. Conversely, in the text discussion, each operation is clearly related to the command used to request it. You can easily find the specifications for that command by turning to the reference specifications, which are arranged in alphabetical order.

HOW TO ENTER COMMANDS

You can enter commands through any standard input device, typically a terminal (for sessions) or a card reader (for jobs). Each command is accepted by the MPE Command Interpreter, which passes it to the appropriate system procedure for execution. Following this execution, control returns to the Command Interpreter, which is now ready for another command. (More information about standard input devices appears on pages 3-1 and 4-2; further discussions about the Command Interpreter appear on pages 3-1 and 4-1.)

COMMAND ELEMENTS

Each MPE command consists of:

- A colon (required in all cases as an MPE command identifier).
- A command name (required in all cases).
- A parameter list (used in most cases).

A typical command including all three elements appears as follows:

:RUN PPOG, ENTRYX

Colon ↑ Command name Parameter list

The *colon* identifies a statement as an MPE command. In an interactive session, MPE prints the colon on the terminal whenever it is ready to accept a command; you respond by entering the remainder of the command after the colon. In a batch job, however, *you* must enter the colon, placing it in column 1 of the source card (or card image) on which the command is to appear.

The *command name*, which you enter immediately after the colon, requests a specific operation. MPE prohibits embedded blanks within the name, and rejects the command if they appear. MPE interprets the next non-alphabetic character encountered as the end of the command name; typically this character is a blank.

The *parameter list* contains one or more parameters that specify options for the command. It is required in some commands, but is optional or prohibited in others. Parameter lists can include positional parameters and/or keyword parameter groups (defined below), separated from each other by delimiters such as commas, semicolons, equal signs, or other punctuation marks.

Normally, you must separate the parameter list from the command name by one or more blanks. (But when you omit the first optional parameter in a positional list, you can begin the list, starting with the comma that normally follows the first parameter, immediately after the command name. The comma serves in place of blanks as a delimiter, as noted under *Positional Parameters*, below.) Within the parameter list, any delimiter can be surrounded by any number of blanks, permitting a free and flexible command format.

MPE permits both decimal and octal numbers as command parameters. You distinguish between the two by preceding the octal numbers with a percent sign (%).

The end of each command is indicated by the end of the record on which it appears — for example, a *carriage return* for terminal input or the end of the card containing the command for card input. But if the last non-blank character of the record is a continuation character (as defined on page 1-5), the command is continued onto the next record.

NOTE

If you are running programs in batch job mode, bear in mind that MPE scans all 80 columns on each card image, and thus *no characters are ignored*.

POSITIONAL PARAMETERS

With positional parameters, the meaning of a parameter depends upon its position in the parameter list. For example, in the :FORTRAN command, issued to compile a FORTRAN program, the parameter list specifies the input file containing the source program, the output file to which the object program is written, and the output file to which the source listing is transmitted, *always in that order*. In the following :FORTRAN command, for instance, the variable names INP, OUT, and *LST indicate the source, object, and list files, respectively.

```
:FORTRAN INP,OUT, *LST
```

NOTE

In *LST, the asterisk (*) is not a delimiter but a special character denoting a back-reference to a previously-defined file, as described on Page 6-22; MPE regards the asterisk as part of the parameter.

Positional parameters are separated from one another by commas. Note that in the above example, the second delimiting comma in the parameter list is followed by an optional blank. In future examples, for clarity, delimiters will always be followed by blanks.

When you omit an optional positional parameter from within a list, you must still include the delimiter that would normally follow that parameter. Thus, on a listing, two adjacent delimiters indicate a missing optional parameter. When you omit a positional parameter that would otherwise immediately follow a command name, indicate this by entering its delimiter as the first character in the parameter list. When you omit positional parameters from the end of the list, however, you need not include delimiters to signify this — the terminating *return* or end-of-card is sufficient. The following examples demonstrate how to properly omit parameters from a command:

```
:FORTRAN , USLFL, *LISTFL, MFL, NFL           First parameter omitted.
:FORTRAN *SOURCEFL, , *LISTFL, MFL, NFL       Second (embedded) parameter omitted.
:FORTRAN *SOURCEFL, USFL, *LISTFL           Last two (trailing) parameters omitted.
:FORTRAN                                     All parameters omitted.
```

KEYWORD PARAMETERS

When a parameter list is so long that use of positional parameters becomes difficult, MPE provides keyword parameter groups. The meaning of such a group is independent of its position in the list — thus, you can enter keyword groups in any order with respect to each other. A keyword group consists of a keyword that denotes the group's meaning, sometimes followed by an equal sign and one or more sub-parameters. Each keyword group is preceded by a semicolon. When more than one sub-parameter appears in a group, they are usually separated from each other by commas. All delimiters can be optionally preceded or followed by blanks. The following example shows a :PREP command containing both positional and keyword parameters. INPT and OUTF are the variable names of the positional parameters. DL and CAP are keywords that designate the keyword parameter groups. PH, DS, and MR are sub-parameters of the keyword group designated by CAP.

```
:PREP INPT, OUTF; DL=500; CAP=PH, DS, MR
```

When both keyword groups and positional parameters form a list, the positional parameters always occur before the keyword groups. When you omit trailing parameters from the positional group in this list, you need not include their delimiters since the occurrence of the first keyword indicates the omission. When you omit optional sub-parameters from a keyword group, simply follow the same rules that apply to positional parameters.

CONTINUATION CHARACTERS

When the length of a command exceeds one record (for instance, one entry-line or source card), you may enter an ampersand (&) as the last non-blank character of the record and continue the command on the next record. This next record must begin with a colon (supplied automatically by MPE in interactive processing, but entered by yourself in batch processing). Optionally, you can embed blanks between the colon that begins the continuation record and the first non-blank character in the continuation record. In the example below, the command contains a continuation character at the end of the first line and an embedded blank at the beginning of the second.

```
:RUN PROGB; NOPRIV; LMAP; STACK=500; PARM=5; &  
: DL=600; LIB=G
```

You can continue commands up to 255 characters; prompting colons and continuation ampersands are not counted as part of this total.

When continuing a command onto another line, you must not divide a command name, keyword, positional parameter, or keyword sub-parameter — MPE does not permit any such element to span more than one line.

MPE does not begin interpretation of a command until the last record of the command is read.

COMMAND ERRORS

If you make an error while entering a command in an interactive session, MPE suppresses execution of that command, displays an error number indicating the type of error (described in Section X), and returns control to your terminal. If you desire a full explanation of the error, you can request it by entering any character other than a carriage-return immediately after the error number. In response, MPE displays the appropriate error message on the next line. If you do not require an explanation of the error, enter a carriage-return directly after the error number. MPE then prints a colon, prompting you for another command.

If you enter an erroneous command in a batch job (and do not precede this command with a :CONTINUE command, as discussed in Section IV), MPE suppresses execution of the command, prints both the appropriate error number and message on your standard list device, ignores all subsequent commands in the job, and aborts the job.

All error numbers and messages generated by the MPE Command Interpreter are discussed in Section X.

EXECUTING COMMANDS PROGRAMMATICALLY

In addition to entering commands directly through your standard input device, MPE allows you to execute many of them from within the programs that you write. You do this by including, within those programs, calls to the COMMAND intrinsic. This intrinsic invokes the Command Interpreter and passes to it command images that MPE will interpret and execute as the corresponding system commands. The command specifications in Section II point out those commands which can be executed programmatically. Complete information on the COMMAND intrinsic appears in *MPE Intrinsic Reference Manual*.

COMMAND LANGUAGE

SECTION

II

The reference specifications for all MPE commands available to standard users appear in this section. For easy reference, they are presented alphabetically by command name; should you prefer to reference them by function, simply remove these pages and re-arrange them according to the *Functional List of Commands* shown in table 2-1. For each command, the reference specifications show the following information:

- Syntax
- Parameter definitions (including meaning, constraints, and defaults).
- When legal (in sessions or jobs, during break, or programmatically).
- Whether interruptable (with BREAK key).
- Operation or functional description.
- Examples
- Where described in text portion of this manual.

In the reference specifications, the indication *Available In Break* means: the command can be entered and executed if the *current* operation is suspended by pressing the BREAK key on the terminal (or calling the CAUSEBREAK intrinsic). The notation *Available Programmatically* means: the command is available through the COMMAND intrinsic. CAUSEBREAK and COMMAND are explained in *MPE Ininsics Reference Manual*. A summary of the rules for entering commands, plus notes on the syntax conventions used in this manual appear in table 2-2.

NOTE

IF YOU HAVE NO PRIOR EXPERIENCE WITH MPE, YOU SHOULD READ THE TEXT DISCUSSION (SECTIONS III THROUGH X) BEFORE ATTEMPTING TO USE THE REFERENCE SPECIFICATIONS IN THIS SECTION.

Table 2-1. Functional List of Commands

FUNCTION	COMMAND
Running Sessions (Section III)	<ul style="list-style-type: none"> :HELLO :BYE :ABORT :RESUME :EOD :EOF:
Running Jobs (Section IV)	<ul style="list-style-type: none"> :JOB :EOJ :STREAM :COMMENT :CONTINUE
Determining Session/Job Status and Resource Usage (Section V)	<ul style="list-style-type: none"> :SHOWJOB :REPORT
Managing Files (Section VI)	<ul style="list-style-type: none"> :FILE :RESET :BUILD :LISTF :RENAME :PURGE :SAVE :STORE :RESTORE :ALTSEC :RELEASE :SECURE :DATA
Running Subsystems and User Programs (Section VII; commands marked with asterisk (*) are discussed in pertinent subsystem manuals.)	<ul style="list-style-type: none"> :BASIC :BASICOMP :BASICPREP :BASICGO :COBOL :COBOLPREP :COBOLGO :FORTRAN :FORTPREP :FORTGO :RPG :RPGPREP :RPGGO :SPL :SPLPREP :SPLGO :PREP :PREPRUN :RUN :DEBUG* :EDITOR* :RJE* :SEGMENTER* :SETDUMP* :RESETDUMP*

Table 2-1. Functional List of Commands (Continued)

FUNCTION	COMMAND
Determining Device and Devicefile Status (Section VIII)	<ul style="list-style-type: none"> :SHOWDEV :SHOWIN :SHOWOUT
Requesting Utility Operations (Section IX)	<ul style="list-style-type: none"> :TELL :TELLOP :SETMSG :SPEED :PTAPE :GETRIN :FREERIN :SHOWTIME

Table 2-2. Reference Notes for Command Definitions

ELEMENTS OF COMMAND FORMAT

- Leading Colon: is the prompt/command identifier character in interactive sessions.
is the command identifier character in batch jobs.
- Command Name: is shown in CAPITAL LETTERS IN REGULAR (ROMAN) TYPE, contains no blanks, is delimited by a non-alphabetic character (usually a blank).
- Parameters: are shown in CAPITAL LETTERS IN REGULAR TYPE when they are literal information that you always enter exactly as shown;
are shown in *lower-case italics* when they are variable parameters to be replaced by information that you must supply.
- Positional Parameters: have significance implied by positional order after command name; use adjacent commas (or semicolons where required) to indicate omitted parameter(s) as follows:
 - :COMMANDNAME p1,,p3 (from middle of list)
 - :COMMANDNAME ,p2,p3 (from beginning of list)
 - :COMMANDNAME p1 (from end of list)
- Keyword Parameters: are separated by semicolons and can appear in any order.
- Mixed Parameters: positional parameters are given first; first keyword indicates end of positional list.
- Optional Parameters:
 - [A] means "A" may be included.
 - $\begin{bmatrix} A \\ B \end{bmatrix}$ means "A" or "B" may be included.
 - $\left. \begin{matrix} A \\ B \end{matrix} \right\}$ means "A" or "B" must be included.
 - $\begin{bmatrix} A \\ B \end{bmatrix}$ means "A" and/or "B" may be included in any order.

USER/SYSTEM DIALOGUE

- User input is underlined where necessary for clarity: NEW NAME? ALPHA1

:ABORT

Aborts current program.

SYNTAX

:ABORT

PARAMETERS

None

USE

Available	In Session?	YES (In Break only)
	In Job?	NO
	In Break?	YES
	Programmatically?	NO
Breakable?		NO

OPERATION

After you suspend a program or MPE command operation by pressing the BREAK key, the :ABORT command immediately terminates that program or operation.

NOTE

The :ABORT command is legitimate *only* during a *Break*.

EXAMPLE

:ABORT

TEXT DISCUSSION

Pages 3-16, 3-18, 3-19.

:ALTSEC

Permanently changes file's security provisions.

SYNTAX

```
:ALTSEC filereference [;([modelist:userlist [; . . ])]
```

PARAMETERS

filereference Actual designator of file whose security provisions are to be altered, written in this format:

filename [/lockword] [.groupname] [.accountname]

The lockword, if any, must be specified. (Required parameter.)

modelist

Modes of access to be permitted to users in *userlist* parameter. The *modelist* is separated from the *userlist* by a colon (:). If two or more modes are specified, they must be separated from each other by commas. The modes are denoted by letters, as follows:

R = Reading
L = Locking (allows exclusive access to file)
A = Appending (implicitly specifies L also)
W = Writing (implicitly specifies A and L also)
X = Executing

If you omit any mode from this list, this implies (at the file level) that no one is permitted access in the omitted mode. If you specify no mode at all, however, this implies (at the file level) completely unrestricted access to the file. Access may, however, be further limited by other provisions specified at the account and group levels. (Required parameter if *userlist* is included.)

userlist

Types of users to whom access modes defined by *modelist* parameter apply. If two or more user types are specified, they must be separated by commas. The user types are specified as follows:

ANY = Any user
AC = Account member
AL = Account librarian user
GU = Group user
GL = Group librarian user
CR = Creating user

(Required parameter if *modelist* is included.)

NOTE

More than one *modelist:userlist* parameter combination can be used, to permit extremely versatile file security specifications. These combinations are separated by semicolons.

If no *modelist:userlist* parameter combination is specified, the following default security provisions are assigned: (R,A,W,L,X: ANY). These, of course, may be overridden by other restrictions imposed at the account or group level.

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	YES
	Programmatically?	YES
Breakable?		NO

:ALTSEC

(Continued)

OPERATION

Permanently deletes all security provisions for a file at file level, and replaces them with provisions specified as command parameters. Does not affect account and group level security provisions or file lockword. This command applies to only one file, not to a collection of files.

NOTE

You can only use this command for a permanent disc file whose label identifies you as the creator of that file. When the normal (default) MPE security provisions are in effect, this must be a file in your log-on account and belonging to your log-on or home group.

Even though you may be barred by *modelist:userlist* restrictions from accessing a file that you have created, you can still issue an :ALTSEC command for that file. Thus, you can change the security provisions to allow yourself access to it.

EXAMPLE

You have created a file named FDATA and you wish to change its security provisions to allow write-access to yourself alone. Enter the following command:

```
:ALTSEC FDATA; (W:CR)
```

To change the security provisions for a program file (FPROG) so that any group user can execute the program but only the account and group librarians can read or write on the file, enter:

```
:ALTSEC FPROG; (X:GU; R,W:AL,GL)
```

TEXT DISCUSSION Page 6-80.

:BASIC

Interprets a BASIC program.

SYNTAX

```
:BASIC [ commandfile ] [, [ inputfile ] [, [ listfile ] ]
```

PARAMETERS

commandfile Actual designator of source file (device) from which BASIC commands and statements are input. Can be any ASCII input file. (Interpreter's formal designator for this file is BASCOM.) If omitted, the default file \$STDINX (current input device) is assigned. (Optional parameter.)

inputfile Actual designator of file containing data input for BASIC program. Can be any ASCII input file. (Interpreter's formal designator for this file is BASIN.) If omitted, the default file \$STDINX (current input device) is assigned. (Optional parameter.)

listfile Actual designator of destination file for program listing and output. Can be any ASCII output file. (Interpreter's formal designator for this file is BASLIST.) If omitted, the default file \$STDLIST (session/job listing device) is assigned. (Optional parameter.)



USE

Available	In Session?	YES
	In Job?	YES
	In Break?	NO
	Programmatically?	NO
Breakable?	YES (Suspended.)	

OPERATION

This command is generally used for on-line programming in BASIC language, but you can also employ it to interpret BASIC programs submitted in batch-job mode. In batch mode, the BASIC >EOD command is required after any data following the BASIC RUN command, or after the RUN command itself if there is no data.

EXAMPLE

To enter commands and data via your standard input device (terminal), with program listing and output also transmitted to the terminal, enter:

```
:BASIC
```

To submit to the BASIC interpreter commands and statements from the command file MYCOMDS, and data from the input file MYDATA, with program listing and output written to the list file MYLIST, enter the following command. (All three files reside on disc.)

```
:BASIC MYCOMDS, MYDATA, MYLIST
```

NOTE

In the above example, no program is executed unless there is a RUN command in the file MYCOMDS. The rules regarding the >EOD command (discussed under OPERATION) also apply to the file MYCOMDS in this example. If the input files are created by the Editor, they must be saved using the UNN option of the Editor's KEEP command.

:BASICGO

Compiles, prepares, and executes a BASIC program.

SYNTAX

```
:BASICGO [ commandfile ] [ ,listfile ]
```

PARAMETERS

commandfile Actual designator of input file from which BASIC compiler commands are read. Can be any ASCII input file. (Compiler's formal designator for this file is BSCTEXT.) If omitted, MPE assigns default designator \$STDINX (current input device). (Optional parameter.)

listfile Actual designator of file on which program listing is written. This can be any ASCII output file. (Compiler's formal designator for this file is BSCLIST.) If omitted, MPE assigns default designator \$STDLIST (typically the terminal in a session or the printer in a batch job). (Optional parameter.)

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	NO
	Programmatically?	NO
Breakable?		YES (Suspended.)

OPERATION

Compiles, prepares, and executes a program from a *SAVE FAST* file (created by BASIC Interpreter). This enables the program to run faster than it would if executed via the Interpreter. When you have written the program, you store it in a *SAVE FAST* file via the BASIC Interpreter command *SAVE filename, FAST*. You can then compile, prepare, and execute the program via the :BASICGO command.

If you do not specify a command file, the BASIC compiler expects your input from your standard session/job input device. If you do not specify a listing file, the BASIC compiler writes your listing to your standard session/job listing device. The BASICGO command compiles into a user subprogram library (USL) file named \$OLDPASS (if this file exists) or \$NEWPASS (if \$OLDPASS does not exist); the USL is not accessible to you, but the resulting program file is available in \$OLDPASS.

EXAMPLE

To compile, prepare, and execute the BASIC program MYPROG, with the listing directed to the disc file LISTFL, enter the following commands. The program file is stored in \$OLDPASS.

```
:BASICGO ,LISTFL           Invokes BASIC compiler.

$CONTROL USLINIT, SOURCE   Initializes USL and requests pro-
                             gram listing.

$COMPILE MYPROG            Compiles program MYPROG.

$EXIT                      Exits from compiler.
```

TEXT DISCUSSION Page 7-15.

:BASICOMP

Compiles a BASIC Program.

SYNTAX

```
:BASICOMP [commandfile] [, [uslfile] [, [listfile]]]
```

PARAMETERS

commandfile Actual designator of input file from which BASIC Compiler commands are read. Can be any ASCII input file. (Compiler's formal designator for this file is BSCTEXT.) If omitted, MPE assigns default designator \$STDINX (current input device). (Optional parameter.)

uslfile Actual designator of user subprogram library (USL) file on which compiled object program is written. Can be any binary output file with *filecode* of USL (or 1024). (Compiler's formal designator for this file is BSCUSL.) If you enter this parameter, it must indicate a file previously created in one of three ways:

1. By saving a USL file (through the :SAVE command) created by a previous compilation where the default value was used for the *uslfile* parameter.
2. By building the USL (through the MPE Segmenter command -BUILDUSL, discussed in *MPE Segmenter Reference Manual*).
3. By creating a new USL file (through the :BUILD command, with the mnemonic USL or decimal code of 1024 used for the *filecode* parameter).

If you omit *uslfile* parameter, MPE assigns as default either \$OLDPASS (if a passed USL file currently exists in session/job) or \$NEWPASS (if no passed file exists). (Optional parameter.)

listfile Actual designator of file on which program listing is written. This can be any ASCII output file. (Compiler's formal designator for this file is BSCLIST.) If omitted, MPE assigns default designator \$STDLIST (typically the terminal in a session or the printer in a batch job). (Optional parameter.)

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	NO
	Programmatically?	NO
Breakable?	YES (Suspended.)	

OPERATION

Compiles a program from a *SAVE FAST* file (created by BASIC Interpreter) onto disc in USL format. This enables the program to run faster than it would if left in the form generated by the Interpreter. When you have written the program, you store it in a *SAVE FAST* file with the BASIC Interpreter command *SAVE filename, FAST*. You can then run the BASIC compiler, compile the program with the BASIC compiler command *COMPILE*, exit from the compiler, prepare the program with the *:PREP* command, and run it via the *:RUN* command.

If you do not specify a command file, the BASIC compiler expects input from your standard session/job input device. If you create the USL prior to compilation, you must assign it a *filecode* of USL (or 1024). If you do not specify a list file, the BASIC compiler transmits the program listing to your standard session/job listing device. On the USL, each program unit exists as an RBM that contains object code plus header information that labels and describes this code. You can compile relocatable binary modules (RBMs) onto the same USL during several compilations.

:BASICOMP

(Continued)

EXAMPLE

To compile the BASIC program MYPROG onto the USL named OBJECT, enter:

:BUILD OBJECT; CODE=USL	Builds USL file.
:BASICOMP ,OBJECT	Invokes BASIC compiler, using commands from \$STDINX and USL named OBJECT.
SCOMPILE MYPROG	Compiles SAVE FAST program called MYPROG.
\$EXIT	Exits from BASIC compiler.

TEXT DISCUSSION Page 7-15.

:BASICPREP

Compiles and prepares a BASIC program.

SYNTAX

```
:BASICPREP [ commandfile] [, [progfile] [, listfile]]
```

PARAMETERS

commandfile

Actual designator of input file from which BASIC compiler commands are read. Can be any ASCII input file. (Compiler's formal designator for this file is BSCTEXT.) If omitted, MPE assigns default designator \$STDINX (current input device). (Optional parameter.)

progfile

Actual designator of program file on which prepared program segments are written. Can be any binary output file with file code of PROG (or 1029). If you enter this parameter, it must indicate a file created in one of two ways:

1. By creating a new program file (through the :BUILD command, with the mnemonic PROG or decimal code of 1029 used for the *filecode* parameter).

NOTE

A program file is limited to only one disc extent. Thus, the *numextents* parameter of the :BUILD command must be 1.

2. By specifying a non-existent file in the *progfile* parameter, in which case a session/job temporary file of the correct size and type is created.

If *progfile* is omitted, the default file \$NEWPASS is assigned. To access this file later, reference it as \$OLDPASS. (Optional parameter.)

listfile

Actual designator of file on which program listing is written. This can be any ASCII output file. (Compiler's formal designator for this file is BSCLIST.) If omitted, MPE assigns default designator \$STDLIST (typically the terminal in a session or the printer in a batch job). (Optional parameter.)

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	NO
	Programmatically?	NO
Breakable?	YES (Suspended.)	

OPERATION

Compiles and prepares a program from a *SAVE FAST* file (created by BASIC Interpreter) into a program file on disc. This enables the program to run considerably faster than it would if executed via the Interpreter.

When you have written the program, you store it in a *SAVE FAST* file via the BASIC Interpreter command *SAVE filename, FAST*. You can then compile/prepare it via the MPE command :BASICPREP and run it via the :RUN command.

If you do not specify a command file, the BASIC compiler expects your input from your standard session/job input device. If you create the program file prior to compilation, you must assign it a *filecode* of PROG (or 1029). If you do not specify a list file, the BASIC compiler transmits the listing output to your session/job listing device. The user subprogram library (USL) file created during compilation is a temporary file passed

:BASICPREP

(Continued)

directly to the preparation mechanism; you can access it under the name \$OLDPASS *only if* the program file is not \$NEWPASS. The program file is also opened as a temporary file. For both files, the Segmenter searches first for a file of the proper name in the session/job temporary file domain; if such a file cannot be found, it then searches for a permanent file of that name. If no program file of the referenced name exists, the Segmenter creates a new program file that is saved in the session/job temporary file domain and prepares into this.

EXAMPLE

To compile and prepare a program from the BASIC SAVE FAST file named MYPROG, with listing directed to the standard listing device, enter:

```
:BASICPREP MYCOMDS
```

The file MYCOMDS is an ASCII file that contains the following BASIC compiler commands:

CONTROL USLINIT, SOURCE	Initializes USL and lists program.
COMPILE MYPROG	Compiles SAVE FAST program.
EXIT	Exits from compiler.

TEXT DISCUSSION Page 7-15.

:BUILD

(Continued)

- F File contains fixed-length records. (Optional parameter.)
- U File contains undefined-length records, with block size equal to logical record size. (Optional parameter.)
- V File contains variable-length records, with block size equal to logical record size on unit-record devices. (Optional parameter.)

NOTE

If F, U, or V is not specified, default value is F.

- BINARY File contains binary-coded records. (Optional parameter.)
- ASCII File contains ASCII-coded records. (Optional parameter.)

NOTE

If neither BINARY nor ASCII is specified, default is BINARY.

- CCTL Indicates you are supplying carriage-control characters with your write requests. Valid for any ASCII file. (Optional parameter.)
- NOCCTL Indicates you are *not* supplying carriage-control characters with your write requests. (Optional parameter.)

NOTE

If neither CCTL nor NOCCTL is specified, default is NOCCTL.

- TEMP Request to create the file as a session/job temporary file, entered in the session/job temporary file directory; when session/job terminates, file is deleted. Default: If TEMP is omitted, file is declared permanent and saved in system file domain. (Optional parameter.)

- device* A *device class name* designating the type of device, or a *logical device number* indicating the specific device, on which the file resides.

The device class name makes a non-specific, generic reference to a *type* of device (such as any disc drive.) This name is assigned to a specific set of devices when the system is configured and initiated. See *System Manager/System Supervisor Manual*. It must contain from one to eight alphanumeric characters, begin with a letter, and terminate with any non-alphanumeric character such as a blank. Examples are: MHDISC, FHDISC.

The logical device number refers to a specific single device. This is a number assigned to each device when the system is generated. This specification is only used when assignment of a particular device is truly necessary.

If the *device* parameter specifies a device class name, all extents are restricted to devices of this class; if the *device* parameter specifies a logical device number, all extents are restricted to the single device identified by that number. Default: Device Class Name DISC. (Optional parameter.)

filecode

Code indicating a specially-formatted file. This code is recorded in the file label and is available to processes accessing the file through the FGETINFO intrinsic. Must be a positive integer ranging from 0 to 1023, or one of the HP-defined integers or mnemonics defined below:

Mnemonic	HP-defined Integer	Defines the File as:
USL	1024	User subprogram library (USL) file.
BASD	1025	BASIC data file.
BASP	1026	BASIC program file.
BASFP	1027	BASIC fast program file.
RL	1028	Relocatable library (RL) file.
PROG	1029	Program file.
STAR	1030	STAR file.
SL	1031	Segmented library (SL) file.
(None)	1040	Cross-Loader ASCII file (SAVE).
(None)	1041	Cross-Loader relocated binary file.
(None)	1042	Cross-Loader ASCII file (DISPLAY).
(None)	1050	EDIT KEEPQ file (non-COBOL).
(None)	1051	EDIT KEEPQ file (COBOL).
(None)	1052	EDIT TEXT file (COBOL).
(None)	1060	RJE punch file.
(None)	1070	QUERY procedure file.

Default: 0. (Optional parameter.)

numrec

Total maximum file capacity, in terms of logical records (for files containing fixed-length records) or blocks (for files containing variable-length and undefined-length records). Maximum file capacity allowed is 2,097,120 sectors. Default: 1023. (Optional parameter.)

numextents

Number of extents (integral number of contiguously-located disc sectors) that can be dynamically allocated to the file as logical records are written to it. The size of each extent (in terms of records) is determined by the *numrecs* parameter value divided by the *numextents* parameter value. Extents can be allocated on any disc in the device class specified in the *device* parameter. If you want to ensure that all extents for a file reside on the same disc, use the logical device number of that disc or a device class name relating to a single disc device, in the *device* parameter.

:BUILD

(Continued)

If specified, *numextents* must be an integer value from 1 to 32. Default: 8. (Optional parameter.)

initialloc

Number of extents to be allocated to the file *at the time it is opened*. Must be an integer from 1 to 32. If attempt to allocate requested space fails, an error message appears. Default: 1. (Optional parameter.)

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	YES
	Programmatically?	YES
Breakable?		NO

OPERATION

Builds a new file on disc and immediately allocates space for it, initialized to blanks (if an ASCII file) or zeros (if a binary file). Unless you specify the TEMP parameter, this is a permanent file.

NOTE

To specify a permanent file, you must have SAVE access to the group to which the new file is to belong. Furthermore, you can only build a file belonging to your log-on account.

This command applies to disc files only.

EXAMPLE

To create a permanent disc file named WORKFILE, with fixed-length records each 80 bytes long, 3 records per block (blockfactor) in ASCII code, enter the following :BUILD command. The file can reside on any disc, has a maximum capacity of 2000 records divided into 10 extents, with 2 extents allocated immediately.

```
:BUILD WORKFILE; REC=-80, 3, F, ASCII; DISC=2000, 10, 2
```

TEXT DISCUSSION Pages 3-15, 6-55, 6-60, 7-5, 7-7, 9-6.

:BYE

Ends an interactive session.

SYNTAX

:BYE

PARAMETERS

None

USE

Available	In Session?	YES
	In Job?	NO
	In Break?	YES
	Programmatically?	NO
Breakable?		YES (Aborts log-off message.)

OPERATION

Terminates a session, resulting in a display of log-off information in the following format:

```
CPU (SEC) = cpusecs
CONNECT (MIN) = connectmin
date, time
END OF SESSION
```

cpusecs Total central-processor time used by the session, in seconds, charged against log-on group and account.

connectmin Total time elapsed between log-on and log-off, rounded upward to nearest minute, charged against log-on group and account.

date Current date (day-of-week, month and day, year)

time Current time (hours:minutes am/pm)

EXAMPLE

To terminate the current session, enter :BYE.

```
:BYE

CPU (SEC) = ?
CONNECT (MIN) = ?
TUE, AUG 26, 1975, 2:02 PM
END OF SESSION
```

TEXT DISCUSSION Pages 3-4, 3-14, 3-22.

:COBOL

Compiles a COBOL program.

SYNTAX

```
:COBOL [textfile] [,uslfile] [,listfile] [,masterfile] [,newfile]]]
```

PARAMETERS

textfile Actual designator of input file from which source program is read. Can be any ASCII input file. (Compiler's formal designator for this file is COBTEXT.) If omitted, MPE assigns default designator \$STDIN (current input device). (Optional parameter.)

uslfile Actual designator of user subprogram library (USL) file on which object program is written. Can be any binary output file with file code of USL (or 1024). (Compiler's formal designator for this file is COBUSL.) If you enter this parameter, it must indicate a file previously created in one of three ways:

1. By saving a USL file (through the :SAVE command) created by a previous compilation where the default value was used for the *uslfile* parameter.
2. By building the USL (through the MPE Segmenter command -BUILDUSL, discussed in *MPE Segmenter Reference Manual*).
3. By creating a new USL file (through the :BUILD command, with the mnemonic USL or decimal code of 1024 used for the *filecode* parameter).

If you omit *uslfile* parameter, MPE assigns as default either \$OLDPASS (if a passed file currently exists in session/job) or \$NEWPASS (if no passed file exists). (Optional parameter.)

listfile Actual designator of file on which program listing is written. This can be any ASCII output file. (Compiler's formal designator for this is file is COBLIST.) If omitted, MPE assigns default designator \$STDLIST (typically the terminal in a session or the printer in a batch job). (Optional parameter.)

masterfile Actual designator of file to be optionally merged with *textfile* and optionally written onto a file named *newfile*, as discussed in the manuals covering compilers. Can be any ASCII input file. (Compiler's formal designator for this file is COBMAST.) If omitted, no merging takes place. (Optional parameter.)

newfile Actual designator of file on which (re-sequenced) records from *textfile* (and *masterfile*) are placed. Can be any ASCII output file. (Compiler's formal designator for this file is COBNEW.) If omitted, no text is written to *newfile*. (Optional parameter.)

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	NO
	Programmatically?	NO
Breakable?		YES (Suspended.)

:COBOL

(Continued)

OPERATION

Compiles source-language program written in COBOL onto a USL file on disc. If you do not specify a source text file, MPE expects input from your standard session/job input device. If you create the USL prior to compilation, you must assign it a *filecode* of USL (or 1024). If you do not specify a list file, MPE transmits the program listing to your standard session/job listing device. On the USL, each program unit exists as a relocatable binary module (RBM) that contains object code plus header information that labels and describes this code. You can compile RBMs onto the same USL during several compilations.

EXAMPLE

To compile a COBOL program that you enter from your session/job input device into an object program in the USL file \$NEWPASS, and write the listing to your standard listing device, enter the following command. (If the next command is one to prepare an object program, \$NEWPASS can be passed to that command as an input file named \$OLDPASS.)

```
:COBOL
```

To compile a COBOL program residing on the disc file SOURCE into an object program on the USL file OBJECT, with a program listing generated on the disc file LISTFL, enter:

```
:BUILD OBJECT; CODE=USL
```

Creates USL File OBJECT in special format.

```
:COBOL SOURCE, OBJECT, LISTFL
```

Compiles program from SOURCE onto OBJECT, creating LISTFL and writing listing to it.

TEXT DISCUSSION

Pages 3-16, 7-4, 7-9.

:COBOLGO

Compiles, prepares, and executes a COBOL Program.

SYNTAX

```
:COBOLGO [textfile] [, [listfile] [, [masterfile] [, [newfile]]]
```

PARAMETERS

<i>textfile</i>	Actual designator of input file from which source program is read. Can be any ASCII input file. (Compiler's formal designator for this file is COBTEXT.) If omitted, MPE assigns default designator \$STDIN (current input device). (Optional parameter.)
<i>listfile</i>	Actual designator of file on which program listing is written. This can be any ASCII output file. (Compiler's formal designator for this file is COBLIST.) If omitted, MPE assigns default designator \$STDLIST (typically the terminal in a session or the printer in a batch job). (Optional parameter.)
<i>masterfile</i>	Actual designator of file to be optionally merged with <i>textfile</i> and optionally written onto a file named <i>newfile</i> , as discussed in the manuals covering compilers. Can be any ASCII input file. (Compiler's formal designator for this file is COBMAST.) If omitted, no merging takes place. (Optional parameter.)
<i>newfile</i>	Actual designator of file on which (re-sequenced) records from <i>textfile</i> (and <i>masterfile</i>) are placed. Can be any ASCII output file. (Compiler's formal designator for this file is COBNEW.) If omitted, no text is written to <i>newfile</i> . (Optional parameter.)

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	NO
	Programmatically?	NO
Breakable?	YES (Suspended.)	

OPERATION

Compiles, prepares, and allocates/executes a COBOL program. If you do not specify a source file, MPE expects your input from your standard session/job input device. If you do not specify a listing file, MPE writes your listing to your standard session/job listing device. This command creates a temporary user subprogram library (USL) file (\$NEW-PASS) that you *cannot* access, and a temporary program file that you *can* access under the name \$OLDPASS.

EXAMPLE

To compile, prepare, and execute a COBOL program entered from your standard input device, with the program listing transmitted to your standard listing device, enter:

```
:COBOLGO
```

To compile, prepare, and execute a COBOL program read from the disc file SOURCE and transmit the resulting program listing to the disc file LISTFL, enter:

```
:COBOLGO SOURCE, LISTFL
```


:COBOLPREP

Compiles and prepares a COBOL program.

SYNTAX

```
:COBOLPREP [textfile] [, [progfile] [, [listfile] [, [masterfile] [, newfile]]]]
```

PARAMETERS

textfile Actual designator of input file from which source program is read. Can be any ASCII input file. (Compiler's formal designator for this file is COBTEXT.) If omitted, MPE assigns default designator \$STDIN (current input device). (Optional parameter.)

progfile Actual designator of program file on which prepared program segments are written. Can be any binary output file with file code of PROG (of 1029). If you enter this parameter, it must indicate a file created in one of two ways:

1. By creating a new program file (through the :BUILD command, with the mnemonic PROG or decimal code of 1029 used for the *filecode* parameter).



NOTE

A program file is limited to only one disc extent. Thus, the *numextents* parameter in the :BUILD command must be 1.

2. By specifying a non-existent file in the *progfile* parameter, in which case a session/job temporary file of the correct size and type is created.

If omitted, the default file \$NEWPASS is assigned. (Optional parameter.)

listfile Actual designator of file on which program listing is written. This can be any ASCII output file. (Compiler's formal designator for this file is COBLIST.) If omitted, MPE assigns default designator \$STDLIST (typically the terminal in a session or the printer in a batch job). (Optional parameter.)

masterfile Actual designator of file to be optionally merged with *textfile* and optionally written onto a file named *newfile*, as discussed in the manuals covering compilers. Can be any ASCII input file. (Compiler's formal designator for this file is COBMAST.) If omitted, no merging takes place. (Optional parameter.)

newfile Actual designator of file on which (re-sequenced) records from *textfile* (and *masterfile*) are placed. Can be any ASCII output file. (Compiler's formal designator for this file is COBNEW.) If omitted, no text is written to *newfile*. (Optional parameter.)

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	NO
	Programmatically?	NO
Breakable?		YES (Suspended.)

:COBOLPREP

(Continued)

OPERATION

Compiles and prepares a COBOL program onto a program file on disc. If you do not specify a source file, MPE expects your input from your standard session/job input device. If you create the program file prior to compilation, you must assign it a *filecode* of PROG (or 1029). If you do not specify a list file, MPE transmits the listing output to your session/job listing device. The user subprogram library (USL) file created during compilation is a temporary file passed directly to the preparation mechanism; you can access it under the name \$OLDPASS *only if* the program file is not \$NEWPASS. The program file is also opened as a temporary file. For both files, the Segmenter searches first for a file of the proper name in the session/job temporary file domain; if such a file cannot be found, it then searches for a permanent file of that name. If no program file of the referenced name exists, the Segmenter creates a new program file that is saved in the session/job temporary file domain and prepares into this.

EXAMPLE

To compile and prepare a COBOL program entered through the session/job input device, onto the file \$NEWPASS, with the listing printed on the session/job listing device, enter the following command. (If the next command is one to execute the program, the file \$NEWPASS is referenced in the execute command under the name \$OLDPASS.)

```
:COBOLPREP
```

To compile and prepare a COBOL source program input from a source file named SFILE into a program file named MYPROG, with the resulting listing generated on the session/job listing device, enter the following command:

```
:COBOLPREP SFILE, MYPROG
```

TEXT DISCUSSION Page 7-6.

:COMMENT

Inserts comment into command stream.

SYNTAX

:COMMENT [<i>text</i>]

PARAMETERS

text

Information comprising comment text. If last non-blank character is ampersand (&), comment text is continued onto next record. Default: Null comment. (A record containing only ":COMMENT" is inserted in command stream.) (Optional parameter.)

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	YES
	Programmatically?	YES
Breakable?		NO

OPERATION

Places a record containing the characters ":COMMENT" followed by text into command stream. Used primarily in batch jobs.

EXAMPLE

The second command in the job below inserts a comment:

```
:JOB MAC.TECHPUBS  
JOB NUMBER = #J3  
WED, SEP 17, 1975, 2:03 PM  
HP32002A.00.00  
  
:COMMENT THIS IS A SAMPLE JOB.
```

TEXT DISCUSSION Page 4-16.

:CONTINUE

Overrides job error.

SYNTAX

:CONTINUE

PARAMETERS

None

USE

Available	In Session?	NO
	In Job?	YES
	In Break?	NO
	Programmatically?	NO
Breakable?		NO

OPERATION

Permits job to continue even though next command results in error (with accompanying error message). Typically used when anticipating errors that you wish to override. Applies *only* to command immediately following :CONTINUE.

EXAMPLES

When you anticipate a possible error arising from the command :RUN MYPROGZ in the job below, and wish to override this error and allow the job to continue, enter the :CONTINUE command as shown:

•
•
•
•

:CONTINUE

Overrides any error in next command.

:RUN MYPROGZ

Attempts to run non-existent program.

ERR 108,1 NON-EXISTENT FILE

Error message from above command.

:RUN MYPROGX

Runs valid program.

VALUE OF A = 1.38
VALUE OF A = .49
VALUE OF A = -.85
END OF PROGRAM

} Output from MYPROGX.

•
•
•

Job continues.

When you attempt to purge a file (:PURGE command) that you are not certain exists, and wish your job to continue in spite of any error resulting from referencing a non-existent file, precede the :PURGE command with the :CONTINUE command:

•
•
•
•
•

:CONTINUE

Overrides error in :PURGE.

:PURGE XYZ

Purges File XYZ, if it exists.

TEXT DISCUSSION Pages 4-20, 4-21, 10-2.

:DATA

Enters data into system from devicefile.

SYNTAX

```
:DATA [sjname,]username[/userpass] .acctname[/acctpass] [;filename]
```

PARAMETERS

(A complete session or job identification.)	<i>sjname</i>	Name of session or job that is to read data. Default: Null session/job name. (Optional parameter.)
	<i>username</i>	Your user name that allows you to access MPE under this account, as established by Account Manager. (Required parameter.)
	<i>userpass</i>	Your user password, optionally assigned by Account Manager. (Required if assigned.)
	<i>acctname</i>	Name of account under which session/job is running, as established by System Manager. (Required parameter.)
	<i>acctpass</i>	Account password, optionally assigned by System Manager. (Required if assigned.)
	<i>filename</i>	Additional qualifying name for the data that can be used by session or job to access the data. May be used, for instance, to distinguish two separate data decks from different cards readers read by same program. If <i>filename</i> is omitted, no such distinguishing name is assigned. (Optional parameter.)

NOTE

The *sjname*, *username*, *userpass*, *acctname*, *acctpass*, and *filename* parameters are all names that can contain up to eight alphanumeric characters, beginning with a letter. Embedded blanks are prohibited in the *username.acctname* entry, and a *period* must be used as a delimiter for this pair of parameters. Passwords must be separated from preceding parameters by slash marks with no surrounding blanks.

USE

Available	In Session?	YES (but not from \$STDIN)
	In Job?	YES (but not from \$STDIN)
	In Break?	NO
	Programmatically?	NO
Breakable?		NO

OPERATION

Identifies input consisting only of data to be read from devicefile other than session/job standard input file. The input device must be one configured by System Supervisor or Console Operator to automatically accept :DATA command. This command is often used to access data from a batch input device while in session mode — for instance, to read a deck from a card reader while running an interactive session. It is *required* for every deck not embedded in standard input stream (\$STDIN) and submitted from a device configured to accept :DATA.

:DATA

(Continued)

NOTE

You cannot use the :DATA command for data submitted from a device *not* configured to accept this command, nor for data embedded within standard input stream nor for data on disc files.

To designate data for your session or job, enter the :DATA command, followed by the data, followed by the :EOD command. The data can only be read by a session/job that has the same identity: [*sjname*,] *username.acctname*.

The *filename* parameter can be used to distinguish between multiple devicefiles directed (via :DATA) to a single session/job. If you omit this parameter, then data can be read by *any* access from session/job with corresponding identity.

If a session/job attempts to read data but omits the *filename* when opening the data file, any file containing a :DATA command referencing that session/job's identity will satisfy the request. If the session/job explicitly references the filename incorrectly, it will not access the data.

When the :DATA file is placed on the input device and the device is readied, MPE reads the entire file (if input from a spooled device) or the :DATA command only (if input from a non-spooled device). From this point on, your session/job can access the data, which remains available until it is actually read.

NOTE

The :DATA command implicitly initiates communication with MPE, and thus is the only command not entered within a formally-initiated session or job. Files identified by :DATA may be input on cards, magnetic or paper tape, or via a terminal. When entered from magnetic tape, such a file must reside on a single tape volume.

EXAMPLE

To designate a data file on cards for a session identified as JOBSP,BLACK.ACTSP, submit the following :DATA command, followed by the data and the terminating :EOD command:

```
:DATA JOBSP,BLACK.ACTSP; FILESP
.
.
.
(YOUR DATA)
.
.
.
:EOD
```

The session can access the data in the card file by specifying the card reader (either by device class name or logical device number) and the file name FILESP.

TEXT DISCUSSION Pages 3-19, 3-22, 4-19, 6-81 through 6-83.

:DEBUG

Invokes MPE Debug Facility

SYNTAX

```
:DEBUG
```

PARAMETERS

None

USE

Available	In Session?	YES
	In Job?	NO
	In Break?	NO
	Programmatically?	YES
Breakable?		NO

NOTE

This command is available to you only if you have the Privileged Mode (PM) Optional Capability, described in *MPE Intrinsic Reference Manual*.

OPERATION

This command is an extension of the MPE Debug Facility and is used primarily by systems programmers. The Debug Facility allows programmers to establish breakpoints within programs from which it is called, and to display and modify data within the stacks or registers used by those programs. Because the :DEBUG command effectively invokes the Debug Facility from the MPE Command Interpreter program, it can establish breakpoints within that program and allow users to operate on the Command Interpreter's stack. The more general applications of the Debug Facility are obtained by calling the DEBUG intrinsic; by using the DEBUG parameter of the :RUN command; or by reaching a previously-established breakpoint. This invokes the Facility from your running program, and allows you to operate on that program's stack; these general applications are discussed in *MPE Debug/Stack Dump Reference Manual*. Once you access this facility, you issue directives to it in response to interactive prompts (in the form of a question mark). These directives and various Debug messages are also discussed in the *MPE Debug/Stack Dump Reference Manual*.

WARNING

When you run Debug in this way, you are operating in *Privileged Mode* and can easily destroy the system if you are not careful. Beware of modifying any locations or setting breakpoints in privileged code.

EXAMPLE

To invoke the Debug Facility from the MPE Command Interpreter, enter the following command (if you have Privileged Mode Capability):

```
:DEBUG
```

TEXT DISCUSSION

MPE Debug/Stack Dump Reference Manual.

SYNTAX

`:EDITOR [listfile]`

PARAMETERS

listfile

Actual designator of file to receive any output resulting from Editor Commands LIST and XPLAIN, specifying OFFLINE option. Can be any ASCII output file. (Editor's formal designator for this file is EDTLIST.) Since this file is often a non-disc file, it is often pre-defined in a :FILE command and back-referenced in the format:

[**listfile*]

If omitted, the Editor assigns a device having device class name LP. (Optional parameter.)

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	NO
	Programmatically?	NO
Breakable?		YES (Suspended.)

OPERATION

Invokes the Editor, used to create and modify ASCII text files. Often used for writing and storing source-program text on-line; resulting file can be saved and input directly to appropriate compiler. The file to be edited is called the TEXT *file*, and is specified after you call the Editor. If you omit the *listfile* parameter, but device class name LP is not defined for your system, an error message is output.

EXAMPLE

To invoke the Editor during a session and specify a disc file named LISTER as the listing device for OFFLINE output, enter:

```
:EDITOR LISTER
```

TEXT DISCUSSION *EDIT/3000 Reference Manual.*

:EOD

Denotes end-of-data on input stream.

SYNTAX

:EOD

PARAMETERS

None

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	NO
	Programmatically?	NO
Breakable?		NO

OPERATION

Denotes the end-of-data read through an input stream. Used primarily to delimit data entered through standard input device. Also used for data submitted through a non-sharable device via the :DATA command.

NOTE

When reading data from the standard input stream (using the file name *\$STDIN*), MPE interprets any record beginning with a colon as the end-of-data, whether or not this record contains an :EOD command. When reading data from the standard input stream (using file name *\$STDINX*), MPE interprets the following entries as indicating the end-of-data: :EOD, :EOF:, :EOJ, :DATA, and :JOB; no other entries or command images (records beginning with a colon in Column 1), however, indicate the end-of-data on this file. In general, most programmers prefer to use :EOD as a data delimiter because it executes no *additional* command functions beyond terminating data.

When encountered by FREAD or READ intrinsic, :EOD causes those intrinsics to return CCG Condition Code to calling program. (See *MPE Intrinsic Reference Manual* for a discussion of condition codes.)

EXAMPLE

:EOD

TEXT DISCUSSION

Pages 3-19, 3-20, 4-2, 4-4, 4-11, 4-12, 6-81 through 6-83, 7-5.

:EOF:

Simulates hardware end-of-file on input stream from any device.

SYNTAX

:EOF:

PARAMETERS

None

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	NO
	Programmatically?	NO
Breakable?		NO

OPERATION

Denotes end-of-file read through an input stream. Required to simulate a physical end-of-file on HP 2894A card reader/punch unit when that device is configured to refuse data submitted via the :DATA command. (This device does not provide a true hardware end-of-file indication.)

EXAMPLE

:EOF:

NOTE

The last colon in this command must be followed by a blank.

TEXT DISCUSSION

Pages 3-22, 4-19.

:EOJ

Ends a batch job.

SYNTAX

:EOJ

PARAMETERS

None

USE

Available	In Session?	NO
	In Job?	YES
	In Break?	NO
	Programmatically?	NO
Breakable?		NO

OPERATION

Terminates a batch job, resulting in a display of log-off information in the following format

```
CPU (SEC) = cpusecs
ELAPSED (MIN) = elapsedmin
date, time
END OF JOB
```

cpusecs Total central-processor time used by the job, in seconds, charged against log-on group and account.

elapsedmin Total time elapsed between beginning and end of job, rounded upward to nearest minute. *Not* charged against log-on group and account.

date Current date (day-of-week, month and day, year).

time Current time (hours:minutes am/pm)

EXAMPLE

To terminate the current job, enter :EOJ as follows:

```
:EOJ

CPU (SEC) = 1
ELAPSED (MIN) = 1
TUE, SEP 16, 1975, 4:14 PM
END OF JOB
```

TEXT DISCUSSION

Pages 4-2, 4-4, 4-10, 4-11, 4-19.

Re-defines file characteristics
supplied by FOPEN intrinsic.

SYNTAX

For New or Old (Existing) Files:

```
:FILE formaldesignator  $\left[ \begin{array}{l} = \$NEWPASS \\ [= filereference] [,NEW] \\ = \$OLDPASS \\ [= filereference] \left[ \begin{array}{l} ,OLD \\ ,OLDTEMP \end{array} \right] \end{array} \right]$ 
```

```
 $\left[ \begin{array}{l} ;REC = \left[ resize \right] \left[ \begin{array}{l} , \left[ blockfactor \right] \left[ \begin{array}{l} \left[ F \\ U \\ V \right] \left[ \begin{array}{l} ,BINARY \\ ,ASCII \end{array} \right] \end{array} \right] \end{array} \right] \end{array} \right]^{\dagger}$ 
```

```
 $\left[ \begin{array}{l} ;CCTL \\ ;NOCCTL \end{array} \right]^{\dagger}$ 
```

```
 $\left[ ;ACC = \left\{ \begin{array}{l} IN \\ OUT \\ UPDATE \\ OUTKEEP \\ APPEND \\ INOUT \end{array} \right\} \right]$ 
```

```
 $\left[ \begin{array}{l} ;NOBUF \\ ;BUF \quad [= numbuffers] \end{array} \right]$ 
```

```
 $\left[ \begin{array}{l} ;EXC \\ ;EAR \\ ;SHR \end{array} \right]$ 
```

```
 $\left[ \begin{array}{l} ;MULTI \\ ;NOMULTI \end{array} \right]$ 
```

```
 $\left[ \begin{array}{l} ;MR \\ ;NOMR \end{array} \right]$ 
```

```
 $\left[ \begin{array}{l} ;DEL \\ ;SAVE \\ ;TEMP \end{array} \right]$ 
```

```
 $\left[ ;DEV=[ device] [, [outputpriority] [, [numcopies]]] \right]^{\dagger}$   
 $\left[ ;CODE = filecode \right]^{\dagger}$   
 $\left[ ;DISC = [numrec] [, [numextents] [, [initalloc]]] \right]^{\dagger}$ 
```

```
 $\left[ \begin{array}{l} ;NOWAIT \\ ;WAIT \end{array} \right]$ 
```

NOTE

The parameter group $\left[;DISC=[*numrec*] [, [*numextents*] [, [*initalloc*]]] \right]$ cannot be included if the parameter group

$\left[[= *filereference*] \left[\begin{array}{l} ,OLD \\ ,OLDTEMP \end{array} \right] \right]$ is specified.

Dagger (†) indicates these parameters are not used for old disc files.

:FILE

(Continued)

SYNTAX (Cont)

For System-Defined Files:

```
:FILE formaldesignator = $NULL
```

or

```
:FILE formaldesignator = { $STDIN  
                          $STDINX  
                          $STDLIST }
```

```
[ ;REC = [ resize ] [ , [ blockfactor ] [ , [ F  
                          U  
                          V ] [ , BINARY  
                                  ASCII ] ] ] ] ]
```

```
[ ;CCTL  
  ;NOCCTL ]
```

```
[ ;ACC = { IN  
          OUT  
          UPDATE  
          OUTKEEP  
          APPEND  
          INOUT } ] ]
```

```
[ ;NOBUF  
  ;BUF [ = numbuffers ] ]
```

```
[ ;EXC  
  ;EAR  
  ;SHR ]
```

```
[ ;MULTI  
  ;NOMULTI ]
```

```
[ ;MR  
  ;NOMR ]
```

```
[ ;NOWAIT  
  ;WAIT ]
```

For User Pre-Defined (Back-Referenced) Files

```
:FILE formaldesignator = *formaldesignator
```

NOTE

The second parameter in this command *must* include the preceding asterisk (*) noted above.

PARAMETERS	<i>formaldesignator</i>	Formal file designator referenced in your program. This is the name by which your program recognizes the file. Contains from 1 to 8 alphanumeric characters, beginning with a letter. (Required parameter.)
	<i>filereference</i>	Actual file designator. This is the name by which MPE recognizes the file, written in the following format: filename[/lockword][.groupname][.accountname] All four sub-parameters are names that contain from 1 to 8 alphanumeric characters, beginning with a letter. If the file has no lockword and belongs to the log-on group and account, only <i>filename</i> is necessary. If you omit the <i>filereference</i> parameter from the :FILE command, the actual designator is equated to the formal designator. (Optional parameter.)
	\$NEWPASS	A temporary disc file that can be automatically passed to any succeeding MPE command within the session/job, which references the file by the name \$OLDPASS. When \$NEWPASS is closed, its name is automatically changed to \$OLDPASS, and any previous file named \$OLDPASS in the session/job is deleted. (Optional parameter.)
	\$OLDPASS	The name of the last temporary disc file closed as \$NEWPASS. (Optional parameter.)

NOTE

If you do not include *any* actual file designator (*filereference*, \$NEWPASS, or \$OLDPASS) in the :FILE command, the formal designator is used as the actual designator.

NEW	Specification that the file is a new file. (Required parameter if <i>filereference</i> is used for a new file; Optional parameter otherwise.)
OLD	Previously-existing permanent file saved in system file domain. File continues to exist after current session/job terminates. (Optional parameter.)
OLDTEMP	Previously-existing temporary file in session/ job temporary file domain. File is deleted at end of current session/job. (Optional parameter.)

NOTE

NEW, OLD, and OLDTEMP specify the file domain indicating where the file exists. If all are omitted, the domain specified in FOPEN intrinsic takes effect.

recsize

Size of logical records in file. If a positive number, this represents *words*; if a negative number, this represents *bytes*. For files containing fixed-length records, this is the size of each logical record. For files containing undefined-length records, this is the maximum record size. For files containing variable-length records, this is a value used to calculate maximum record size as follows:

$$\text{maxrecordsize} = \text{recsize} \times \text{blockfactor}$$

Default: Determined by System Supervisor during System Configuration. The values generally specified by HP are:

Disc	128
Tape	128
Printer	66
Card Reader	40
Card Punch	40
Terminal	40

(Optional parameter.)

blockfactor

Size of each buffer established for file, specified as an integer equal to number of logical records per block. For fixed-length records, *blockfactor* is actual number of records in a block. For variable-length records, *blockfactor* is a multiplier used to compute block size (maximum *recsize* x *blockfactor*). For undefined length records, *blockfactor* is always one logical record per block; any unused portion of the block is filled with ASCII blanks or binary zeros. The blockfactor value may be overridden by MPE. Default: Calculated by dividing the specified *recsize* into the configured blocksize; this value is rounded downward to an integer that is never less than 1. (Optional parameter.)

F

File contains fixed-length records. (Optional parameter.)

U

File contains undefined-length records, with block size equal to logical record size. (Optional parameter.)

V

File contains variable-length records, with block size equal to logical record size on unit-record devices. (Optional parameter.)

NOTE

If F, U, or V is not specified, the default value is F for disc and magnetic tape files, and U for all others.

BINARY

File contains binary-coded records. (Optional parameter.)

ASCII

File contains ASCII-coded records. (Optional parameter.)

NOTE

If neither BINARY nor ASCII is specified, the default value is BINARY.

CCTL Indicates you are supplying carriage-control characters with your write requests. Valid for any ASCII file. (Optional parameter.)

NOCCTL Indicates you are *not* supplying carriage-control characters with your write requests. (Optional parameter.)



NOTE

If neither CCTL nor NOCCTL are specified and carriage-control characters are pertinent to the file, the default value is NOCCTL.

IN File permits read-access only. (FWRITE, FUPDATE, and FWRITEDIR intrinsics cannot reference this file.) (Optional parameter.)

OUT File permits write-access only. This option re-sets end-of-file indicator to first record. Any data on the file prior to current FOPEN request is deleted. (FREAD, FREADSEEK, and FREADDIR intrinsics cannot reference this file.) (Optional parameter.)

OUTKEEP File permits write-access only, allowing you to add new records both before and after current end-of-file indicator. (FREAD, FREADSEEK, and FREADDIR intrinsics cannot reference this file.) (Optional parameter.)

APPEND File permits append-access only, allowing you to add new records after current end-of-file indicator only. (FREAD, FREADSEEK, FREADDIR, FPOINT, FSPACE, and FWRITEDIR intrinsics cannot reference this file.) (Optional parameter.)

INOUT File permits input/output access. (Any file intrinsic except FUPDATE can be issued against this file.) (Optional parameter.)

UPDATE Update access. (All file intrinsics can be issued for this file.) (Optional parameter.)

NOTE

IN, OUT, UPDATE, OUTKEEP, APPEND, and INOUT all belong to the ACC = keyword group. If this entire group is omitted in the :FILE command (and in the FOPEN intrinsic that opens the file), the default value assigned is IN for files on all devices except those intended for output-only such as card punches, printers, paper tape punches, and plotters. If a process attempts to violate these access restrictions on a file, an error is returned. (Optional parameter.)

NOBUF Specification that no input/output buffering is to take place, and no buffers are allocated for the file. (Optional parameter.)

numbuffers Number of buffers to be allocated for the file. This must be an integer, with a maximum value of 16. If set to 0, a value of 2 is assigned. Do not use this parameter for files representing interactive terminals, since a system-managed buffering method is used in such cases. (Optional parameter.)

NOTE

If *NOBUF* and *numbuffers* are both omitted, a default value of 2 buffers is assigned.

EXC After file is opened, prohibits concurrent access (in any mode) to file through another FOPEN request, whether issued by this or another running program (process), until this process issues FCLOSE or terminates. (Optional parameter.)

EAR After file is opened, prohibits concurrent write-access to file through another FOPEN request within this or another process, until this process issues an FCLOSE or terminates. (Optional parameter.)

SHR After file is opened, permits concurrent access to file through another FOPEN request issued by this or another process in any session or job. (Optional parameter.)

NOTE

If AC=IN is specified, and EXC, EAR, and SHR are omitted, then default assigned is SHR. Otherwise, default assigned is EXC.

MULTI Shares this file so that several concurrently-running programs (processes) *in this session/job* can transfer records to and from the file sequentially, regardless of *blockfactor* or buffering (if any) specified. All access-control information, including buffers themselves, is shared between the processes, with result that transfers occur in the order in which the processes request access. You cannot, however, share the file in this way with processes running under other sessions/jobs. (Optional parameter.)

NOMULTI Prohibits sharing files in MULTI mode. (Optional parameter.)

NOTE

If MULTI and NOMULTI are both omitted, the default assigned is NOMULTI.

MR Request for Multi-record Mode. Individual read or write requests are not confined to record boundaries. Thus, if the number of words or bytes to be transferred exceeds the size of the record, the remaining words or bytes are taken from subsequent records until the number requested are transferred. Restricted to NOBUF files. (Optional parameter.)

NOMR Request for normal recording mode (with no multi-record read/write). Individual read/write requests are confined to record boundaries. (Optional parameter.)

NOTE

If MR and NOMR are both omitted, the default assigned is NOMR.

DEL	Request for regular temporary file disposition; the file is deleted when your program closes it. (Optional parameter.)
SAVE	Request for permanent file disposition; the file remains in the system file domain, potentially available to other users, when your program closes it. (Optional parameter.)
TEMP	Request for session/job temporary file disposition. The file remains in the session/job temporary file domain when your program closes it; but when your session/job terminates, the file is deleted. (Optional parameter.)

NOTE

DEL, SAVE, and TEMP denote the disposition of the file. If none of these keywords is specified in this :FILE command, or the disposition is not specified in the FCLOSE intrinsic that closes the file, the file is returned to the disposition it had when opened. For example, a new file (other than \$NEWPASS) is deleted; an old file is returned to the domain in which it was found.

device

A *device class name* designating the type of device, or a *logical device number* indicating the specific device, on which the file resides.

The device class name makes a non-specific, generic reference to a *type* of device (such as any disc drive or magnetic tape unit). This name is defined and related to a specific set of devices when the system is configured. It must contain from one to eight alphanumeric characters, begin with a letter, and terminate with any non-alphanumeric character such as a blank. Examples are CARD, LP, TTY, and TAPE2.

The logical device number refers to a specific single device. This is a number assigned to each device when the system is generated. This specification is only used when assignment of a particular device is truly necessary. For example, you would specify the logical device number of a specific device when running a hardware diagnostic program for that device.

The device specification is *not* used if the file is an old disc file or if the actual file designator used is \$STDIN, \$STDINX, \$STDLIST, \$NEWPASS, \$OLDPASS, or \$NULL (since these names are already assigned to devices by the system).

If the file is a new disc file and the *device* parameter specifies a device class name, all extents are restricted to devices of this class; if the *device* parameter specifies a logical device number, all extents are restricted to the single device identified by that number. Default: Device Class Name DISC. (Optional parameter.)

outputpriority (Spooled output devicefiles only.) The output priority for this file, used to select next spooled devicefile (on disc) for output, among all those contending for a specific printer or card punch. Determines the order in which files are produced when several are awaiting the same device. This must be a value between 1 (lowest priority) and 13 (highest priority), inclusive. If this value is less than the current *outfence* set by the Console Operator, the file is deferred from printing/punching until the operator raises the *outputpriority* of the file or lowers the *outfence*. This parameter is ignored for non-spooled output devices. Default: 8 (if logging enabled) or 13 (if logging disabled). (Optional parameter.)

numcopies (Spooled output devicefiles only.) Number of copies of file to be produced by spooling facility. The copies will not appear contiguously if the Console Operator intervenes or if a file of equal or higher *outputpriority* becomes READY before the last copy is started. This parameter is ignored for non-spooled output devices. Default: 1. (Optional parameter.)

filecode Code indicating a specially-formatted file. This code is recorded in the file label and is available to programs accessing the file through the FGETINFO intrinsic. Must be a positive integer ranging from 0 to 1023, or one of the HP-defined integers or mnemonics listed below:

Mnemonic	HP-defined Integer	Defines the File as:
USL	1024	User subprogram library (USL) file.
BASD	1025	BASIC data file.
BASP	1026	BASIC program file.
BASFP	1027	BASIC fast program file.
RL	1028	Relocatable library (RL) file.
PROG	1029	Program file.
STAR	1030	STAR file.
SL	1031	Segmented library (SL) file.
(None)	1040	Cross-Loader ASCII file (SAVE).
(None)	1041	Cross-Loader relocated binary file.
(None)	1042	Cross-Loader ASCII file (DISPLAY).
(None)	1050	EDIT KEEPQ file (non-COBOL).
(None)	1051	EDIT KEEPQ file (COBOL).
(None)	1052	EDIT TEXT file (COBOL).
(None)	1060	RJE punch file.
(None)	1070	QUERY procedure file.

Default: 0. (Optional parameter.)

numrec Total maximum file capacity, specified only for a NEW file, in terms of logical records (for files containing fixed-length records) and blocks (for files containing variable-length and undefined-length records). Maximum capacity allowed is 2,097,120 sectors. Default: 1023. (Optional parameter.)

numextents Number of extents (integral number of contiguously-located disc sectors) that can be dynamically allocated to the file as logical records are written to it; specified for new files only. The size of each extent (in terms of records) is determined by the *numrecs* parameter value divided by the *numextents* parameter value. Extents can be allocated on any disc in the device class specified in the *device* parameter. If you want to ensure that all extents for a file reside on the same disc, use the logical device number of that disc or a device class name relating to a single disc device, in the *device* parameter.

If specified, *numextents* must be an integer value from 1 to 32. Default: 8. (Optional parameter.)

initialloc Number of extents to be allocated to the file *at the time it is opened*, specified for new files only. Must be an integer from 1 to 32. If attempt to allocate requested space fails, the FOPEN intrinsic that opens the file returns an error condition code to your program. Default: 1. (Optional parameter.)

NOWAIT Allows program that references this file to initiate an input/output request and to have control returned before completion of the request. To confirm completion of such requests, your program must call the IOWAIT intrinsic after each request, as described in *MPE Intrinsic Reference Manual*. Also, multi-record access is prohibited.

NOTE

To use the NOWAIT parameter, your program must be running in Privileged Mode. If you specify NOWAIT, this implies that no buffering (NOBUF) and normal recording mode (NOMR) are used. (Optional parameter.)

WAIT Disallows NOWAIT input/output, discussed above. (Optional parameter.)

NOTE

If NOWAIT and WAIT are both omitted, WAIT is assigned by default.

**formaldesignator* Name of a file defined in a previous :FILE command, preceded by an asterisk. (Required parameter.)

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	YES
	Programmatically?	YES
Breakable?		NO

:FILE

(Continued)

OPERATION

Re-specifies characteristics for a file. Used to override FOPEN intrinsic parameters (or their default values supplied by MPE). Does not, however, override label on old disc file or physical constraints of device on which file resides. Takes effect when subsystem or program opens the file referenced; remains in effect throughout your session or job, unless superseded by a later :FILE command or revoked by a :RESET command. At session/job termination, the :FILE command is cancelled. Applies to files on any devices.

NOTE

A :FILE command deletes *all* specifications appearing in any previous :FILE command that references the same file.

EXAMPLES

A program references two files by the file names (formal file designators) SOURCE and DEST, but you wish to use two disc files recognized in the system by the actual file designators INX and OUTX respectively in place of these files. You can re-specify the file names when you run your program in this fashion:

```
→ :FILE SOURCE = INX
→ :FILE DEST = OUTX
:RUN MYPROG
```

You run a program that reads input from a file it calls INPT and directs output to a file called OUTPT. INPT is a card file (opened as an old file with 80-byte logical records and device class name CARD). OUTPT is a new file on disc, to be assigned an actual file designator FILEX with 64-word, fixed-length logical records written as two records per block in ASCII code. The file may contain a maximum of 800 logical records, is immediately allocated two extents, and may have a maximum of ten extents. When your program closes the file, it is saved as a permanent file in the system file domain. You can specify these files as follows:

```
→ :FILE INPT, OLD; REC = -80; DEV = CARD
→ :FILE OUTPT = FILEX, NEW; REC = 64,2,F,ASCII; DISC=800,10,2; SAVE
:RUN PROGFLX
```

TEXT DISCUSSION: Pages 6-13 through 6-54, 7-7, 7-8, 7-10.

:FORTGO

Compiles, prepares, and executes a FORTRAN program.

SYNTAX

```
:FORTGO [textfile] [, [listfile] [, [masterfile] [, [newfile]]]
```

PARAMETERS

<i>textfile</i>	Actual designator of input file from which source program is read. Can be any ASCII input file. (Compiler's formal designator for this file is FTNTEXT.) If omitted, MPE assigns default designator \$STDIN (current input device). (Optional parameter.)
<i>listfile</i>	Actual designator of file on which program listing is written. This can be any ASCII output file. (Compiler's formal designator for this file is FTNLIST.) If omitted, MPE assigns default designator \$STDLIST (typically the terminal in a session or the printer in a batch job). (Optional parameter.)
<i>masterfile</i>	Actual designator of file to be optionally merged with <i>textfile</i> and optionally written onto a file named <i>newfile</i> , as discussed in the manuals covering compilers. Can be any ASCII input file. (Compiler's formal designator for this file is FTNMAST.) If omitted, no merging takes place. (Optional parameter.)
<i>newfile</i>	Actual designator of file on which (re-sequenced) records from <i>textfile</i> (and <i>masterfile</i>) are placed. Can be any ASCII output file. (Compiler's formal designator for this file is FTNNEW.) If omitted, no text is written to <i>newfile</i> . (Optional parameter.)

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	NO
	Programmatically?	NO
Breakable?	YES (Suspended.)	

OPERATION

Compiles, prepares, and allocates/executes a FORTRAN program. If you do not specify a source file, MPE expects your input from your standard session/job input device. If you do not specify a listing file, MPE writes your listing to your standard session/job listing device. This command creates a temporary user subprogram library (USL) file (\$NEW-PASS) that you *cannot* access, and a temporary program file that you *can* access under the name \$OLDPASS.

EXAMPLE

To compile, prepare, and execute a FORTRAN program entered from your standard input device, with the program listing transmitted to your standard listing device, enter:

```
:FORTGO
```

To compile, prepare, and execute a FORTRAN program entered from the disc file SOURCE and transmit the resulting program listing to the disc file LISTFL, enter:

```
:FORTGO SOURCE, LISTFL
```

TEXT DISCUSSION Pages 3-3, 4-2, 4-3, 7-8, 9-9.

:FORTPREP

Compiles and prepares a FORTRAN program.

SYNTAX

```
:FORTPREP [textfile] [, [progrfile] [, [listfile] [, [masterfile] [, [newfile]]]]
```

PARAMETERS

textfile Actual designator of input file from which source program is read. Can be any ASCII input file. (Compiler's formal designator for this file is FTNTEXT.) If omitted, MPE assigns default designator \$STDIN (current input device). (Optional parameter.)

progrfile Actual designator of program file on which prepared program segments are written. Can be any binary output file with file code of PROG (or 1029). If you enter this parameter, it must indicate a file created in one of two ways:

1. By creating a new program file (through the :BUILD command, with the mnemonic PROG or decimal code of 1029 used for the *filecode* parameter).

NOTE

A program file is limited to only one disc extent. Thus, the *numextents* parameter of the :BUILD command must be 1.

2. By specifying a non-existent file in the *progrfile* parameter, in which case a session/job temporary file of the correct size and type is created.

If omitted, the default file \$NEWPASS is assigned. (Optional parameter.)

listfile Actual designator of file on which program listing is written. This can be any ASCII output file. (Compiler's formal designator for this file is FTNLIST.) If omitted, MPE assigns default designator \$STDLIST (typically the terminal in a session or the printer in a batch job). (Optional parameter.)

masterfile Actual designator of file to be optionally merged with *textfile* and optionally written onto a file named *newfile*, as discussed in the manuals covering compilers. Can be any ASCII input file. (Compiler's formal designator for this file is FTNMAST.) If omitted, no merging takes place. (Optional parameter.)

newfile Actual designator of file on which (re-sequenced) records from *textfile* (and *masterfile*) are placed. Can be any ASCII output file. (Compiler's formal designator for this file is FTNNEW.) If omitted, no text is written to *newfile*. (Optional parameter.)

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	NO
	Programmatically?	NO
Breakable?		YES (Suspended.)

:FORTPREP

(Continued)

OPERATION

Compiles and prepares a FORTRAN program onto a program file on disc. If you do not specify a source file, MPE expects your input from your standard session/job input device. If you create the program file prior to compilation, you must assign it a *filecode* of PROG (or 1029). If you do not specify a list file, MPE transmits the listing output to your session/job listing device. The user subprogram library (USL) file created during compilation is a temporary file passed directly to the preparation mechanism; you can access it under the name \$OLDPASS *only if* the program file is not \$NEWPASS. The program file is also opened as a temporary file. For both files, the Segmenter searches first for a file of the proper name in the session/job temporary file domain; if such a file cannot be found, it then searches for a permanent file of that name. If no program file of the referenced name exists, the Segmenter creates a new program file that is saved in the session/job temporary file domain and prepares into this.

EXAMPLE

To compile and prepare a FORTRAN program entered through the session/job input device, onto the file \$NEWPASS, with the listing printed on the session/job listing device, enter the following command. (If the next command is one to execute the program, the file \$NEWPASS is referenced in the execute command under the name \$OLDPASS.)

```
:FORTPREP
```

To compile and prepare a FORTRAN source program input from a source file named SFILE into a program file named MYPROG, with the resulting listing generated on the session/job listing device, enter the following command:

```
:FORTPREP SFILE, MYPROG
```

TEXT DISCUSSION Pages 7-6, 7-7.

:FORTRAN

Compiles a FORTRAN program.

SYNTAX

```
:FORTRAN [textfile] [, [uslfile] [, [listfile] [, [masterfile] [, [newfile]]]]
```

PARAMETERS

- textfile* Actual designator of input file from which source program is read. Can be any ASCII input file. (Compiler's formal designator for this file is FTNTEXT.) If omitted, MPE assigns default designator \$STDIN (current input device). (Optional parameter.)
- uslfile* Actual designator of user subprogram library (USL) file on which object program is written. Can be any binary output file with file code of USL (or 1024). (Compiler's formal designator for this file is FTNUSL.) If you enter this parameter, it must indicate a file previously created in one of three ways:
1. By saving a USL file (through the :SAVE command) created by a previous compilation where the default value was used for the *uslfile* parameter.
 2. By building the USL (through the MPE Segmenter command -BUILDUSL, discussed in *MPE Segmenter Reference Manual*.)
 3. By creating a new USL file (through the :BUILD command, with the mnemonic USL or decimal code of 1024 used for the *filecode* parameter).
- If you omit *uslfile* parameter, MPE assigns as default either \$OLDPASS (if a passed file currently exists in session/job) or \$NEWPASS (if no passed file exists). (Optional parameter.)
- listfile* Actual designator of file on which program listing is written. This can be any ASCII output file. (Compiler's formal designator for this file is FTNLIST.) If omitted, MPE assigns default designator \$STDLIST (typically the terminal in a session or the printer in a batch job). (Optional parameter.)
- masterfile* Actual designator of file to be optionally merged with *textfile* and optionally written onto a file named *newfile*, as discussed in the manuals covering compilers. Can be any ASCII input file. (Compiler's formal designator for this file is FTNMAST.) If omitted, no merging takes place. (Optional parameter.)
- newfile* Actual designator of file on which (re-sequenced) records from *textfile* (and *masterfile*) are placed. Can be any ASCII output file. (Compiler's formal designator for this file is FTNNEW.) If omitted, no text is written to *newfile*. (Optional parameter.)

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	NO
	Programmatically?	NO
Breakable?		YES (Suspended.)

:FORTRAN

(Continued)

OPERATION

Compiles source-language program written in FORTRAN onto a USL file on disc. If you do not specify a source text file, MPE expects input from your standard session/job input device. If you create the USL prior to compilation, you must assign it a *filecode* of USL (or 1024). If you do not specify a list file, MPE transmits the program listing to your standard session/job listing device. On the USL, each program unit exists as a relocatable binary module (RBM) that contains object code plus header information that labels and describes this code. You can compile RBMs onto the same USL during several compilations.

EXAMPLE

To compile a FORTRAN program that you enter from your session/job input device into an object program in the USL file \$NEWPASS, and write the listing to your standard listing device, enter the following command. (If the next command is one to prepare an object program, \$NEWPASS can be passed to that command as an input file named \$OLDPASS.)

```
:FORTRAN
```

To compile a FORTRAN program residing on the disc file SOURCE into an object program on the USL file OBJECT, with a program listing generated on the disc file LISTFL, enter:

```
:BUILD OBJECT; CODE=USL
```

Creates USL File OBJECT in special format.

```
:FORTRAN SOURCE, OBJECT, LISTFL
```

Compiles program from SOURCE onto OBJECT, creating LISTFL and writing listing to it.

TEXT DISCUSSION Pages 3-16, 7-5, 7-7, 7-16.

:FREERIN

Releases a global RIN.

SYNTAX

```
:FREERIN rin
```

PARAMETERS

rin The RIN to be released. Must be a number from 1 to 1024.
(Required Parameter.)

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	NO
	Programmatically?	NO
Breakable?		NO

OPERATION

If you have acquired a global Resource Identification Number (RIN) with the :GETRIN command, you can release it by issuing :FREERIN. This returns the RIN to the RIN pool managed by MPE. (A RIN is used to manage a resource shared between two or more sessions/jobs so that only one session/job at a time can access that resource.)

NOTE

To free a RIN, you must be the original owner of that RIN — the user who actually issued the :GETRIN command that allocated the RIN and assigned it a password.

EXAMPLE

To release RIN 1, enter

```
:FREERIN 1
```

TEXT DISCUSSION

Pages 9-10, 9-11.

:GETRIN

Acquires global RIN and assigns password for it.

SYNTAX

```
:GETRIN rinpassword
```

PARAMETERS

rinpassword

Password required in the intrinsic that locks the RIN. This is a string of up to eight alphanumeric characters, beginning with a letter. Must be unique. (Required parameter.)

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	YES
	Programmatically?	YES
Breakable?		NO

OPERATION

This command acquires a Global Resource Identification Number (RIN). (A RIN is used in managing a resource shared between two or more sessions/jobs so that only one session/job at a time can access this resource, as described in *MPE Intrinsic Reference Manual*.) MPE displays the RIN on your standard list device as follows:

RIN: *rin*

rin The RIN acquired—a number ranging from 1 to 1024.

Users who know the RIN and its password can use the RIN in future sessions and jobs until the user who issued :GETRIN releases the RIN by entering the :FREERIN command.

The RIN allocated is always a positive integer unique within MPE. The total number of RINs that MPE can allocate is specified when the system is configured, and in no case can exceed 1024.

EXAMPLE

To acquire a global RIN and assign it the password MYRIN, enter the following command:

```
:GETRIN MYRIN  
PIN: 1
```

TEXT DISCUSSION

Pages 9-10, 9-11.

:HELLO

Begins an interactive session.

SYNTAX

```
:HELLO [sessionname,]username[/userpasw].acctname[/acctpaw]
        [,groupname[/grouppaw]]

        [;TERM = termtype]
        [;TIME = cpusecs]

        [;PRI = { BS
                  CS
                  DS
                  ES } ]

        [;INPRI = inputpriority ]
        [;HIPRI ]
```

PARAMETERS

sessionname Arbitrary name used in conjunction with *username* and *acctname* parameters to form a fully-qualified session identity. Contains from 1 to 8 alphanumeric characters, beginning with a letter. Default: null session name. (Optional parameter.)

NOTE

A fully-qualified session identity consists of:

[*sessionname*,]*username*.*acctname*

and furnishes the minimum information required for log-on. Embedded blanks are forbidden in *username*.*acctname* combination.

username A user name, established by Account Manager, that allows you to log-on under this account. This name is unique within the account. Contains from 1 to 8 alphanumeric characters, beginning with letter. (Required parameter.)

userpasw Your user password, optionally assigned by Account Manager. Contains from 1 to 8 alphanumeric characters, beginning with letter. Separated from *username* by slash with no surrounding blanks, as in *username/userpasw*. (Required if assigned.)

acctname Name of your account, as established by System Manager. Contains from 1 to 8 alphanumeric characters, beginning with letter.

NOTE

Must be preceded by period as a delimiter.

(Required parameter.)

acctpaw Account's password, optionally assigned by System Manager. Contains from 1 to 8 alphanumeric characters, beginning with letter. Separated from *acctname* by slash with no surrounding blanks, as in *acctname/acctpaw*. (Required if assigned.)

:HELLO

(Continued)

<i>groupname</i>	Name of file group to be used for local file domain and central-processor time charges, as established by Account Manager. Contains from 1 to 8 alphanumeric characters, beginning with letter. Default: Your home group if you are assigned one by Account Manager. (<i>Optional</i> if you have a home group; <i>Required</i> if you do not.)																				
<i>groupnasw</i>	Group's password, optionally assigned by Account Manager. Contains from 1 to 8 alphanumeric characters, beginning with a letter. Separated from <i>groupname</i> by slash with no surrounding blanks, as in <i>groupname/groupnasw</i> . (Not needed when you log-on under home group. Otherwise, required if assigned.)																				
<i>termtype</i>	Type of terminal used for input. (MPE uses this parameter to determine device-dependent characteristics such as delay factors for carriage returns.) Must be number from 0 to 11, as follows: <table><tr><td>0</td><td>ASR 33 EIA-compatible HP 2749B (10 characters per second (cps)).</td></tr><tr><td>1</td><td>ASR 37 EIA-compatible (10 cps).</td></tr><tr><td>2</td><td>ASR 35 EIA-compatible (10 cps).</td></tr><tr><td>3</td><td>Execuport 300 Data Communication Transceiver Terminal (10, 15, 30 cps).</td></tr><tr><td>4</td><td>HP 2600A or DATAPOINT 3300 (10-240 cps).</td></tr><tr><td>5</td><td>Memorex 1240 (10, 15, 30, 60 cps).</td></tr><tr><td>6</td><td>GE Terminet 300 or 1200, or Data Communication Terminal, Model B (10, 15, 30, 120 cps).</td></tr><tr><td>9</td><td>HP 2615A (MiniBee) (10-240 cps).</td></tr><tr><td>10</td><td>HP 2640A or HP 2644 Interactive Display Terminal (when used predominantly in character mode.) (10-240 cps).</td></tr><tr><td>11</td><td>HP 2640A or HP 2644 Interactive Display Terminal. (Recommended if used at speeds exceeding 30 cps when you expect transfers in block/line mode.) (10-240 cps).</td></tr></table>	0	ASR 33 EIA-compatible HP 2749B (10 characters per second (cps)).	1	ASR 37 EIA-compatible (10 cps).	2	ASR 35 EIA-compatible (10 cps).	3	Execuport 300 Data Communication Transceiver Terminal (10, 15, 30 cps).	4	HP 2600A or DATAPOINT 3300 (10-240 cps).	5	Memorex 1240 (10, 15, 30, 60 cps).	6	GE Terminet 300 or 1200, or Data Communication Terminal, Model B (10, 15, 30, 120 cps).	9	HP 2615A (MiniBee) (10-240 cps).	10	HP 2640A or HP 2644 Interactive Display Terminal (when used predominantly in character mode.) (10-240 cps).	11	HP 2640A or HP 2644 Interactive Display Terminal. (Recommended if used at speeds exceeding 30 cps when you expect transfers in block/line mode.) (10-240 cps).
0	ASR 33 EIA-compatible HP 2749B (10 characters per second (cps)).																				
1	ASR 37 EIA-compatible (10 cps).																				
2	ASR 35 EIA-compatible (10 cps).																				
3	Execuport 300 Data Communication Transceiver Terminal (10, 15, 30 cps).																				
4	HP 2600A or DATAPOINT 3300 (10-240 cps).																				
5	Memorex 1240 (10, 15, 30, 60 cps).																				
6	GE Terminet 300 or 1200, or Data Communication Terminal, Model B (10, 15, 30, 120 cps).																				
9	HP 2615A (MiniBee) (10-240 cps).																				
10	HP 2640A or HP 2644 Interactive Display Terminal (when used predominantly in character mode.) (10-240 cps).																				
11	HP 2640A or HP 2644 Interactive Display Terminal. (Recommended if used at speeds exceeding 30 cps when you expect transfers in block/line mode.) (10-240 cps).																				

Additional information on terminals, including characteristics peculiar to the HP 2640 or 2644 when operating in block versus character mode, appears in Section III.

Default: For hardwired terminals, determined by System Supervisor during system configuration; for terminals that are *not* hardwired, MPE attempts to handle the input/output in the best manner possible based on available information. (Required parameter to ensure correct listings if your terminal is *not* the default *termtype*.)

cpusecs Maximum central-processor time that your session can use, entered in seconds. When this limit is reached, session is aborted. Must be value from 1 to 32767. To specify no limit, enter question mark or omit this parameter.

Default: no limit. (Optional parameter.)

BS }
CS }
DS }
ES }

The execution priority class that the Command Interpreter uses for your session, and also the default priority for all programs executed within the session. BS is highest priority; ES is lowest. If you specify a priority that exceeds the highest permitted for your account or user name by the system, MPE assigns the highest priority possible below BS. Default: CS.

NOTE

DS and ES are intended primarily for batch jobs; their use for sessions is generally discouraged.

(Optional parameter.)

inputpriority

Relative input priority used in checking against access restrictions imposed by the *job fence*, if one exists. Takes effect at log-on time. Must be a value from 1 (lowest priority) to 13 (highest priority). If you supply a value less than or equal to current job fence set by Console Operator, session is denied access.

Default: 8 if logging of session/job initiation is enabled, 13 otherwise. (Optional parameter.)

HIPRI

Request for maximum session-selection input priority, causing session to be scheduled regardless of current job fence or execution limit for sessions.

NOTE

You can specify this parameter only if you have System Manager or Supervisor Capability.

(Optional parameter.)

USE

Available	In Session?	YES
	In Job?	NO
	In Break?	NO
	Programmatically?	NO
Breakable?	YES (Log-on message aborted.)	

:HELLO

(Continued)

OPERATION

Initiates an interactive session by establishing contact with MPE. From this point on, you are automatically prompted by the appearance of a colon whenever MPE is ready to accept a command during your session.

NOTE

Your must enter the :HELLO command via a terminal — no other device can be used for this purpose.

If you omit any passwords required in the :HELLO command, MPE prompts you for each of them individually.

When MPE initiates the session, it transmits the following information to your terminal:

```
SESSION NUMBER = #Snnnn  
date, time  
HP 32002v.uu.ff
```

nnnnn Session number assigned by MPE to uniquely identify the session. This may range from one to four digits long.

date Current date (day-of-week, month and day, year)

time Current time (hours:minutes am/pm)

v MPE version level. (A letter.)

uu MPE update level. (A number.)

ff MPE fix level. (A number.)

EXAMPLE

To start a session named ALPHA, under the user name MAC, account name TECHPUBS, and file group XGROUP (requiring Password XPASS), enter the following :HELLO command. (Log-on information printed by MPE follows this command.)

```
:HELLO ALPHA,MAC,TECHPUBS,XGROUP/XPASS  
SESSION NUMBER = #S241  
TUE, AUG 26, 1975, 1:59 PM  
HP32002A.00.00
```

TEXT DISCUSSION

Pages 3-1, 3-2, 3-4 through 3-14, 3-22.

SYNTAX

```
.JOB [jobname,]username[/userpasw].acctname[/acctpasw] [,groupname
  [/grouppasw]]
```

```
[;TERM = termtype]
```

```
[;TIME = cpusecs]
```

```
[;PRI = { BS
          CS
          DS
          ES } ]
```



```
[;INPRI = inputpriority ]
[;HIPRI
```

```
[;RESTART]
```

```
[;OUTCLASS = [device] [, [outputpriority] [, numcopies]]]
```

PARAMETERS

jobname Arbitrary name used in conjunction with *username* and *acctname* parameters to form a fully-qualified job identity. Contains from 1 to 8 alphanumeric characters, beginning with a letter. Default: null job name. (Optional parameter.)

NOTE

A fully-qualified job identity consists of:

```
[jobname,]username.acctname
```

and furnishes the minimum information required for access to MPE. Embedded blanks are forbidden in *username.acctname* combination.

username A user name, established by Account Manager, that allows you to access MPE under this account. This name is unique within the account. Contains from 1 to 8 alphanumeric characters, beginning with letter. (Required parameter.)

userpasw Your user password, optionally assigned by Account Manager. Contains from 1 to 8 alphanumeric characters, beginning with letter. Separated from *username* by slash with no surrounding blanks, as in *user/userpasw*. (Required if assigned.)

acctname Name of your account, as established by System Manager. Contains from 1 to 8 alphanumeric characters, beginning with letter.

NOTE

Must be preceded by period as a delimiter.

(Required parameter.)

acctpasw Account's password, optionally assigned by System Manager. Contains from 1 to 8 alphanumeric characters, beginning with letter. Separated from *acctname* by slash with no surrounding blanks, as in *acctname/acctpasw*. (Required if assigned.)

<i>groupname</i>	Name of file group to be used for local file domain and central-processor time charges, as established by Account Manager. Contains from 1 to 8 alphanumeric characters, beginning with letter. Default: Your home group if you are assigned one by Account Manager. (Optional if you have a home group; required if you do not.)																				
<i>grouppasw</i>	Group's password, optionally assigned by Account Manager. Contains from 1 to 8 alphanumeric characters, beginning with letter. Separated from <i>groupname</i> by slash with no surrounding blanks, as in <i>groupname/grouppasw</i> . (Not needed to access MPE under home group. Otherwise, required if assigned.)																				
<i>termtype</i>	Type of terminal used for input. (MPE uses this parameter to determine device-dependent characteristics such as delay factors for carriage returns.) Must be number from 0 to 11, as follows: <table><tr><td>0</td><td>ASR 33 EIA-compatible HP 2749B (10 characters per second (cps)).</td></tr><tr><td>1</td><td>ASR 37 EIA-compatible (10 cps).</td></tr><tr><td>2</td><td>ASR 35 EIA-compatible (10 cps).</td></tr><tr><td>3</td><td>Execuport 300 Data Communication Transceiver Terminal (10, 15, 30 cps).</td></tr><tr><td>4</td><td>HP 2600A or DATAPOINT 3300 (10-240 cps).</td></tr><tr><td>5</td><td>Memorex 1240 (10, 15, 30, 60 cps).</td></tr><tr><td>6</td><td>GE Terminet 300 or 1200, or Data Communication Terminal, Model B (10, 15, 30, 120 cps).</td></tr><tr><td>9</td><td>HP 2615A (MiniBee) (10-240) cps).</td></tr><tr><td>10</td><td>HP 2640A or HP 2644 Interactive Display Terminal (when used predominantly in character mode.) (10-240 cps).</td></tr><tr><td>11</td><td>HP 2640A or HP 2644 Interactive Display Terminal. (Recommended if used at speeds exceeding 30 cps when you expect transfers in block/line mode.) (10-240 cps).</td></tr></table>	0	ASR 33 EIA-compatible HP 2749B (10 characters per second (cps)).	1	ASR 37 EIA-compatible (10 cps).	2	ASR 35 EIA-compatible (10 cps).	3	Execuport 300 Data Communication Transceiver Terminal (10, 15, 30 cps).	4	HP 2600A or DATAPOINT 3300 (10-240 cps).	5	Memorex 1240 (10, 15, 30, 60 cps).	6	GE Terminet 300 or 1200, or Data Communication Terminal, Model B (10, 15, 30, 120 cps).	9	HP 2615A (MiniBee) (10-240) cps).	10	HP 2640A or HP 2644 Interactive Display Terminal (when used predominantly in character mode.) (10-240 cps).	11	HP 2640A or HP 2644 Interactive Display Terminal. (Recommended if used at speeds exceeding 30 cps when you expect transfers in block/line mode.) (10-240 cps).
0	ASR 33 EIA-compatible HP 2749B (10 characters per second (cps)).																				
1	ASR 37 EIA-compatible (10 cps).																				
2	ASR 35 EIA-compatible (10 cps).																				
3	Execuport 300 Data Communication Transceiver Terminal (10, 15, 30 cps).																				
4	HP 2600A or DATAPOINT 3300 (10-240 cps).																				
5	Memorex 1240 (10, 15, 30, 60 cps).																				
6	GE Terminet 300 or 1200, or Data Communication Terminal, Model B (10, 15, 30, 120 cps).																				
9	HP 2615A (MiniBee) (10-240) cps).																				
10	HP 2640A or HP 2644 Interactive Display Terminal (when used predominantly in character mode.) (10-240 cps).																				
11	HP 2640A or HP 2644 Interactive Display Terminal. (Recommended if used at speeds exceeding 30 cps when you expect transfers in block/line mode.) (10-240 cps).																				

Additional information on terminals, including characteristics peculiar to the HP 2640 or 2644 when operating in block versus character mode, appears in Section III.

Default: For hardwired terminals, determined by System Supervisor during system configuration; for terminals that are *not* hardwired, MPE attempts to handle the input/output in the best manner possible based on available information. (Required parameter to ensure correct listings if your terminal is *not* the default *termtype*.)

cpusecs Maximum central-processor time allowed your job, in seconds. When this limit is reached, job is aborted. Must be value from 1 to 32767. To specify no limit, enter question mark or omit this parameter.

Default: no limit. (Optional parameter.)

BS
CS
DS
ES

The execution priority class that the Command Interpreter uses for your job, and also the default priority for all programs executed within the job. BS is highest priority; ES is lowest. If you specify a priority that exceeds the highest permitted for your account or user name by the system, MPE assigns the highest priority possible below BS. Default: DS (unless CS is specified by System Supervisor via the :JOBPRI command).

inputpriority

Relative input priority used in checking against access restrictions imposed by job fence (if one exists). Takes effect when job is entered. Must be value from 1 (lowest priority) to 13 (highest priority). If value is less than or equal to job fence set by Console Operator, job is deferred. Default: 8 (when logging of session/job initiation is enabled) or 13 (when logging is disabled). (Optional parameter.)

HIPRI

Request for maximum input priority, causing job to be scheduled regardless of current job fence or execution limit for jobs. Overrides *inputpriority*.

NOTE

You can specify this parameter only if you have System Manager or Supervisor Capability. (See *System Manager/System Supervisor Manual*.)

(Optional parameter.)

RESTART

Request to re-start a spooled job interrupted by system termination/restart. Takes effect automatically when system is subsequently restarted with the WARMSTART option.

This parameter applies only to jobs initiated on spooled input devices, and is ignored for other jobs. (Optional parameter.)

device

Class name or logical device number of device to receive listing output. Cannot specify a magnetic tape unit. Default: standard listing device assigned by system. (Optional parameter.)

NOTE

To use the *device* parameter, you must have the non-sharable device (ND) file access attribute. (See *System Manager/System Supervisor Manual*.)

:JOB

(Continued)

outputpriority

Output priority for job list file, if destined for spooled line printer or card punch. Used to select next spooled devicefile (on disc) for output, among all those contending for a specific printer or punch. Must be value from 1 (lowest priority) to 13 (highest priority).

NOTE

When output priority is 1, output is always deferred. Thus, you should use an output priority of 2 or greater if you wish output written from disc.

This parameter applies only to output destined for spooled output devices, and is ignored for other output. Default: 8 (when session/job initiation logging is on) or 13 (otherwise). (Optional parameter.)

numcopies

Number of copies of job listing file to be produced. This parameter applies only when listing is directed to a spooled device, and is ignored in other cases. Must be value from 1 to 255. Default: 1. (Optional parameter.)

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	NO
	Programmatically?	NO
Breakable?		NO

OPERATION

Initiates a batch job by establishing contact with MPE. When MPE begins the job, it transmits the following information to the listing device:

JOB NUMBER = #Jnnnn

date,time

HP 32002*v.uu.ff*

nnnn Job number assigned by MPE to uniquely identify job to the system. May range from one to four digits long.

date Current date (day-of-week, month and day, year)

time Current time (hours:minutes am/pm)

v MPE version level. (Letter.)

uu MPE update level. (Number.)

ff MPE fix level. (Number.)

EXAMPLE:

To start a job named BETA, under the user name MAC, account named TECHPUBS, and file group XGROUP (requiring Password XPASS), enter the following :JOB command. (Log-on information printed by MPE follows this command.)

```
:JOB BETA,MAC.TECHPUBS,XGROUP/XPASS  
JOB NUMBER = #J1  
TUE, SEP 16, 1975, 4:14 PM  
HP32002A.00.00
```

NOTE

All subsequent MPE commands must begin with a colon (:), explicitly entered by yourself.

TEXT DISCUSSION: Pages 4-2 through 4-9, 4-19, 4-20.

:LISTF

Lists descriptions of one or more permanent disc files.

SYNTAX

$$:\text{LISTF} \left[\begin{array}{l} \text{file} [\text{.group} [\text{.account}]] \\ @ [\text{.group} [\text{.account}]] \\ @ [\text{.account}] \end{array} \right] \left[, \left[\begin{array}{l} 0 \\ 1 \\ 2 \\ -1 \end{array} \right] \right] [; \text{listfile}]$$

PARAMETERS

<i>file.group.account</i>	List file named in group and account designated.
<i>file.group</i>	List file named in group designated under log-on account.
<i>file</i>	List file named under log-on group.
<i>@.group.account</i>	List all files in group named under account designated.
<i>@.group</i>	List all files in group named under log-on account.
<i>@</i>	List all files in log-on group.
<i>@.@.account</i>	List all files in all groups under account named.
<i>@.@</i>	List all files in all groups under log-on account.
<i>@.@.@</i>	List all files in system.

NOTE

All of the above are optional parameters; if all are omitted, MPE lists all files in your log-on group.

0	Display file name only. (An asterisk suffix indicates the file is open.)
1	Display file name, plus file code, record size (W indicates words, B indicates bytes), record format (F, U, or V), whether ASCII (A) or binary (B) records, whether carriage-control option is taken (C if so), current end-of-file location, and maximum number of records allowed in file.
2	Display detail listed by Option 1, plus blocking factor, number of disc sectors in use (including those for file label and user headers), number of extents currently allocated, maximum number of extents allowed.
-1	Display octal listing of file label.

NOTE

You can only request this option if you have System Manager or Account Manager Capability. If you have only Account Manager Capability, you can only request this option for files in your account.

NOTE

All of the above are optional parameters; if all are omitted, MPE selects Option 0 by default. A specification greater than 2 defaults to 2. A specification less than -1 defaults to -1.

:LISTF

(Continued)

listfile

Name of output file on which description is written. Automatically specified as new ASCII file with variable-length records, user-supplied carriage-control characters (CCTL), OUT access mode, and EXC (exclusive access) option. All other characteristics are same as :FILE command default specifications. If omitted, \$STDLIST is assigned by default.

NOTE

If you specify the -1 option (above) and direct listing output to a non-spooled device (such as a magnetic tape unit) that enters the NOT READY state, then the System Directory will be locked down and the system may hang up. Avoid this type of specification if possible. (Optional parameter.)

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	YES
	Programmatically?	YES
Breakable?		YES (Aborted.)

OPERATION

Lists descriptions of one or more disc files at level of detail you select. You need not have access to a file to list the description of it. You may list Level 0, 1, and 2 information for any file in the system. In addition, if you have Account Manager Capability, you can list Level -1 data for files in your account. If you have System Manager Capability, you can list Level -1 data for any file in the system.

NOTE

This command applies to permanent disc files in the system file domain only.

The output may appear in the following formats:

- Level 0

FILENAME

MYFILE MYFILE1 MYFILE2

- Level 1

ACCOUNT= TECHPUBS GROUP= PUB

FILENAME	CODE	-----LOGICAL RECORD-----			
		SIZE	TYP	EOF	LIMIT
MYFILE		128W	FB	0	1023
MYFILE1		128W	FB	0	1023
MYFILE2		128W	FB	0	1023

:LISTF

(Continued)

- Level 2

ACCOUNT= TECHPUBS GROUP= PUB

FILENAME	CODE	-----LOGICAL RECORD-----				----SPACE----		ACC
		SIZE	TYP	EOF	LIMIT R/B	SECTORS #	MX	
MYFILE		128W	FB	0	1023	1	128	1 8

- Level -1

F= MYFILE

046531	043111	046015	020040	001000	152041				MYFILE.....
046531	043111	046105	020040	050125	041040	020040	020040		MYFILE..PUB.....
052105	041510	050125	041123	046501	041440	020040	020040		TECHPUBSMAC.....
020040	020040	020040	020040	020202	004040	000001	113535	
113535	113535	000000	000000	001400	000000	000000	001777	
000000	000000	011663	000000	002001	177400	000200	000407	
000000	000200	000000	000000	000000	152041	000000	000000	
000000	000000	000000	000000	000000	000000	000000	000000	
000000	000000	000000	000000	000000	000000	000000	000000	
000000	000000	000000	000000	000000	000000	000000	000000	
000000	000000	000000	000000	000000	000000	000000	000000	
000000	000000	000000	000000	000000	000000	000000	000000	
000000	000000	000000	000000	000000	000000	000000	000000	
000000	000000	000000	000000	000000	000000	000000	000000	
000000	000000	000000	000000	000000	000000	000000	000000	
000000	000000	000000	000000	000000	000000	000000	000000	
000000	000000	000000	000000	000000	000000	000000	000000	
000000	000000	000000	000000	000000	000000	000000	000000	

EXAMPLE

To list the names of all files in your log-on group, enter:

```
:LISTF
```

To list the name, file code, record format, ASCII vs. binary code information, carriage-control option, end-of-file location, and maximum number of records for the file named BASE in the group USERGP under your log-on account, enter:

```
:LISTF BASE.USERGP, 1
```

To list the names of all files in all groups in your log-on account, enter:

```
:LISTF *.*
```

NOTE

For more detailed examples and listings, see description of the LISTDIR2 program in *MPE System Utilities Manual*.

Prepares program from a USL file onto a program file.

SYNTAX

```
:PREP uslfile,progfile
      [;ZERODB]
      [;PMAP]
      [;MAXDATA=segsz]
      [;STACK=stacksz]
      [;DL=dlsz]
      [;CAP=caplist]
      [;RL=filename]
```

PARAMETERS

uslfile Actual designator of user subprogram library (USL) file on which program has been compiled. (Required parameter.)

progfile Actual designator of program file onto which prepared program segments are to be written. Can be any binary output file. You must create this program file in either of two ways:

1. By creating a new file of program-file type (through the :BUILD command, with the decimal code of 1029 or mnemonic PROG used for *filecode* parameter).

NOTE

A program file is limited to only one disc extent. Thus, the *numextents* parameter of the :BUILD command must be 1.

2. By specifying a non-existent file in *progfile* parameter of :PREP command (or in *progfile* parameter of Segmenter subsystem command -PREPARE, as discussed in *MPE Segmenter Reference Manual*), in which case a file of the correct size and type is created. This file is a session/job temporary file. (Required parameter.)

ZERODB Request to initialize to zero initially-defined user-managed (DL-DB) area, and uninitialized portions of the DB-Q (initial) area. If this parameter is omitted, these areas are not affected. (Optional parameter.)

PMAP Request to produce a descriptive listing of the prepared program on file whose formal designator is SEGLIST; if no :FILE command referencing SEGLIST is encountered, listing is produced on session/job listing device. If this parameter is omitted, listing is not produced. (Optional parameter.)

segsz Maximum stack area (Z-DL) size permitted, *in words*. This parameter is included if you expect to change size of DL-DB or Z-DB areas during program execution. If omitted, MPE assumes that you will not change these areas. (Optional parameter.)

stacksz Size of your initial local data area (Z-Q (initial)) in stack, *in words*. (This value *must* exceed 511 words.) This overrides *stacksz* estimated by Segmenter, which applies if *stacksz* parameter is omitted. (Default is a function of estimated stack requirements for each program unit in program. Since it is difficult for system to predict behavior of stack at run time, you may want to override default by supplying your own estimate with *stacksz*.) (Optional parameter.)

:PREP

(Continued)

dlsize DL-DB area to be initially assigned to stack. This area is of interest mainly in programmatic applications. In all cases, the DL-DB area is rounded upward so that the distance from the beginning of the stack data segment to the DB-address is a multiple of 128 words. If *dlsize* parameter is omitted, a value estimated by Segmenter applies. (Optional parameter.)

caplist *Capability-class attributes* associated with your program, specified as two-character mnemonics. If more than one mnemonic is specified, each must be separated from its neighbor by a comma.

The mnemonics are:

IA = Interactive access	}	Standard Capabilities
BA = Local batch access		
PH = Process handling		
DS = Data segment management		
MR = Multiple resource management		
PM = Privileged-mode operation		

When you issue the :PREP command, you can only specify capabilities that you yourself possess (through assignment by the System or Account Manager). If you do not specify any capabilities, the IA and BA capabilities (if you possess them) are assigned to this program; if you do not possess either or both of these capabilities, that capability is not assigned. (Optional parameter.)

filename Actual designator of relocatable procedure library (RL) file to be searched to satisfy external references during preparation, as described in *MPE Segmenter Reference Manual*. This can be any permanent binary file of type RL. It need not belong to log-on group, nor does it have a reserved, local name. This file yields a single segment that is incorporated into the segments of the program file. If *filename* is omitted, no library is searched. (Optional parameter.)

NOTE

For the *stacksize*, *dlsize*, and *segsiz*e parameters, a value of -1 indicates that the Segmenter is to assign the default value; it is equivalent to omitting the parameter.

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	NO
	Programmatically?	NO
Breakable?		YES (Suspended.)

OPERATION

Prepares a compiled program onto a program file, ready for execution. Unless you explicitly create this file before invoking :PREP, this program file will be created by this command as a session/job temporary file. This command permits you to request searching of a relocatable library (RL) to satisfy references to external procedures required by your program. Within such libraries, these procedures exist in relocatable binary module (RBM) form. When program is prepared, these procedures are prepared into a single segment and linked to the program in the resulting program file.

EXAMPLE

To prepare a program from the USL file named USEFILE and store it in a program file named PROGFILE, enter the command shown below. The optional parameters are those assigned by default. The program's capability-class attributes are the standard capabilities that you possess.

```
:PREP USEFILE, PROGFILE
```

To accomplish the same function as the previous command, but to also list the prepared program, assign a stacksize of 500 words, and assign batch-access capability only for the program, enter:

```
:PREP USEFILE, PROGFILE; PMAP; STACK=500; CAP=BA
```

TEXT DISCUSSION Pages 7-3, 7-4, 7-7, 7-9, 7-10, 7-19.

:PREPRUN

Prepares and executes a compiled program.

SYNTAX

```
:PREPRUN uslfile [,entrypoint]  
    [;NOPRIV]  
    [;PMAP]  
    [;DEBUG]  
    [;LMAP]  
    [;ZERODB]  
    [;MAXDATA=segsz]  
    [;PARM=parameternum]  
    [;STACK=stacksize]  
    [;DL=dlsz]  
  
    [;LIB= { G  
            P  
            S } ]  
  
    [;CAP=caplist]  
    [;RL=filename]  
    [;NOCB]
```

PARAMETERS

<i>uslfile</i>	Actual designator of user subprogram library (USL) file on which program has been compiled. (Required parameter.)
<i>entrypoint</i>	Program entry-point where execution is to begin. May be primary entry point of program, or any secondary entry point in program's outer block. If parameter is omitted, primary entry point is assigned by default. (Optional parameter.)
NOPRIV	Declaration that program segments will be placed in <i>non-privileged</i> (user) mode. This parameter is intended for programs prepared with privileged-mode capability. Normally, program segments containing privileged instructions are executed (run) in privileged mode <i>only</i> if program was <i>prepared</i> with privileged-mode (PM) capability-class. (A program containing legally-compiled privileged code, placed in non-privileged mode, may abort when an attempt is made to execute it.) If NOPRIV is specified in :PREPRUN command, all program segments are placed in <i>non-privileged</i> mode. (Library segments are not affected since their mode is determined independently.) (Optional parameter.)
PMAP	Request to produce a descriptive listing of the prepared program on file whose formal designator is SEGLIST; if no :FILE command referencing SEGLIST is encountered, listing is produced on session/job listing device. If this parameter is omitted, listing is not produced. (Optional parameter.)
DEBUG	A request to set a debug breakpoint on the first executable instruction of the program. If omitted, no breakpoint is set. (Optional parameter.)
LMAP	Request to produce a descriptive listing of the allocated (loaded) program on file whose formal designator is LOADLIST; if no :FILE command referencing LOADLIST is encountered, listing is produced on session/job listing device. If LMAP is omitted, listing is not produced. (Optional parameter.)
ZERODB	Request to initialize to zero initially-defined user-managed (DL-DB) area, and uninitialized portions of the DB-Q (initial) area. If this parameter is omitted, these areas are not affected. (Optional parameter.)

:PREPRUN

(Continued)

<i>segsiz</i>	Maximum stack area (Z-DL) size permitted, <i>in words</i> . This parameter is included if you expect to change size of DL-DB or Z-DB areas during program execution. If omitted, MPE assumes that you will not change these areas. (Optional parameter.)
<i>parameternum</i>	Value that can be passed to your program as a general parameter for control or other purposes; when program is executed, this value can be retrieved from address Q(initial)-4 where Q(initial) is Q-address for outer block of program. Value can be octal number or signed or unsigned decimal number. If <i>parameternum</i> is omitted, Q(initial)-4 address is filled with zeros. (Optional parameter.)
<i>stacksize</i>	Size of your initial local data area (Z-Q(initial)) in stack, in <i>words</i> . (This value <i>must</i> exceed 511 words.) This overrides <i>stacksize</i> estimated by Segmenter, which applies if <i>stacksize</i> parameter is omitted. (Default is a function of estimated stack requirements for each program unit in program. Since it is difficult for system to predict behavior of stack at run time, you may want to override default by supplying your own estimate with <i>stacksize</i> .) (Optional parameter.)
<i>dlsiz</i>	DL-DB area to be initially assigned to stack. This area is of interest mainly in programmatic applications. In all cases, the DL-DB area is rounded upward so that the distance from the beginning of the stack data segment to the DB-address is a multiple of 128 words. If <i>dlsiz</i> parameter is omitted, a value estimated by Segmenter applies. (Optional parameter.)
G	Search segmented procedure libraries in the following order to satisfy external references during allocation: Group library, followed by account public library, followed by system library.
P	Search segmented procedure libraries in the following order to satisfy external references during allocation: Account public library followed by system library.
S	Search segmented procedure libraries in the following order to satisfy external references during allocation: System library only.

NOTE

If G, P, and S are all omitted, S is assumed. (Optional parameters.)

caplist *Capability-class attributes* associated with your program, specified as two character mnemonics. If more than one mnemonic is specified, each must be separated from its neighbor by a comma.

The mnemonics are:

IA = Interactive access	}	Standard Capabilities
BA = Local batch access		
PH = Process handling		
DS = Data segment management		
MR = Multiple resource management		
PM = Privileged-mode operation		

:PREPRUN

(Continued)

When you issue the :PREPRUN command, you can only specify capabilities that you yourself possess (through assignment by the System or Account Manager). If you do not specify any capabilities, the IA and BA capabilities (if you possess them) are assigned to this program; if you do not possess either or both of these capabilities, that capability is not assigned. (Optional parameter.)

filename Actual designator of relocatable procedure library (RL) file to be searched to satisfy external references during preparation, as described in *MPE Segmenter Reference Manual*. This can be any permanent binary file of type RL. It need not belong to log-on group, nor does it have a reserved, local name. This file yields a single segment that is incorporated into the segments of the program file. If *filename* is omitted, no library is searched. (Optional parameter.)

NOCB Request that file system not use stack segment (PCBX) for its control blocks, even if sufficient space is available. This will permit you to expand your stack (via the **DLSIZE** or **ZSIZE** intrinsics) to the maximum possible limit at a later time, but will cause the File Management System to operate more slowly for this program.

NOTE

You should only use this parameter if program absolutely requires largest stack possible.

NOTE

For the *stacksize*, *dlsiz*, and *segsiz* parameters, a value of -1 indicates that the Segmenter is to assign the default value; it is equivalent to omitting the parameter.

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	NO
	Programmatically?	NO
Breakable?	YES (Suspended.)	

OPERATION

Prepares and executes a program compiled in a USL. This command prepares a temporary program file and then executes the program in that file. (The Segmenter creates \$NEWPASS and prepares into it, and the Loader then executes \$OLDPASS.) This command permits you to specify searching of both relocatable (RL) and segmented (SL) libraries to satisfy external references.

:PREPRUN

(Continued)

EXAMPLE

To prepare and execute a program on the USL file USEF, with no special parameters declared, enter the following command. (All default values apply.)

```
:PREPRUN USEF
```

To prepare and execute a program on the USL file UBASE, beginning execution at the entry-point RESTART, declaring a stacksize of 800 words, and specifying that the library LIBA will be searched to satisfy external references, enter:

```
:PREPRUN UBASE, RESTART; STACK=800; RL=LIBA
```

TEXT DISCUSSION Pages 7-3, 7-4, 7-11, 7-12, 7-19.

:PTAPE

Reads paper tape without X-OFF Control.

SYNTAX


`:PTAPE filename`

PARAMETERS

filename Name of existing ASCII file on disc, to which input data is written from a paper tape. (This is normally a file with variable-length records; the record size specified must be great enough to contain the longest paper tape record.) (Required parameter.)

USE

Available	In Session?	YES
	In Job?	NO
	In Break?	YES
	Programmatically?	YES
Breakable?		NO



OPERATION

Permits programs to read input originating from a terminal coupled to a paper-tape reader/punch, when the tape does not include the X-OFF control character punched on tapes created by MPE. (The X-OFF character, when encountered on a tape, turns the tape-reading mechanism off. This character normally appears following the carriage-return/line-feed character combination used to delimit each record.) The :PTAPE command transfers all data from the tape to an ASCII file on disc, named in the *filename* parameter. This operation stores all records onto the disc file in the order read from the tape, but deletes all carriage-return and line-feed characters. (If the tape contains X-OFF characters, the operation also deletes these. But the X-OFF characters are *not* ignored by the tape reader, and so you must re-start the reader after each X-OFF is read.) The input terminates when the Y^c (CONTROL-Y) character is encountered on the tape or entered from the terminal, returning control to the keyboard. The input data can now be read from the disc file by any program.

NOTE

The :PTAPE command is required **ONLY** for input from HP tape readers associated with terminal keyboards.

You cannot use the :PTAPE command to copy tapes containing more than 32,767 bytes of information. If you attempt to do so, MPE reads only the first 32,767 bytes on the tape and then terminates the operation.

EXAMPLE

To copy input from a paper tape onto a disc file named TAPEFILE, create an appropriate file (with the :BUILD command) and then enter the :PTAPE command shown:

```
:BUILD TAPEFILE; REC=-80,3,V,ASCII  
:PTAPE TAPEFILE
```

TEXT DISCUSSION Pages 9-6 through 9-10.

:PURGE

Deletes file from system.

SYNTAX

```
:PURGE filereference [,TEMP]
```

PARAMETERS

filereference Actual file designator of file to be deleted, in this format:

```
filename[ /lockword] [.groupname] [.accountname]
```

To delete the file, you must have write (W) access to it, as defined in the MPE security provisions. (Required parameter.)

TEMP

Indicates that the file is a temporary file in the session/job temporary file domain. You *must* enter this parameter to delete a temporary file; otherwise, the appropriate domain is not searched for the file. If omitted, permanent file is assumed. (Optional parameter.)

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	YES
	Programmatically?	YES
Breakable?		NO

OPERATION

Deletes a disc file from the system, removing it from current domain and directory.

NOTE

This command applies to permanent and temporary files on disc only.

If the file does not exist in the domain specified, the following message appears:

FILE NOT FOUND

EXAMPLE

To delete the permanent file PFILE, enter:

```
:PURGE PFILE
```

To delete the temporary file TFILE, enter:

```
:PURGE TFILE, TEMP
```

If you attempt to purge a temporary file (XYK) but do not include the TEMP parameter, the *FILE NOT FOUND* message appears because the permanent file domain was searched (in vain):

```
:PURGE XYK  
FILE NOT FOUND
```

TEXT DISCUSSION Pages 3-15, 6-62, 6-63.

:RELEASE

Temporarily suspends all security provisions for a file.

SYNTAX

```
:RELEASE filereference
```

PARAMETERS

filereference

Actual designator of file whose security provisions are to be suspended, written in this format:

filename[/lockword] [.groupname] [.accountname]

If the file has a lockword, you must specify it. (Required parameter.)

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	YES
	Programmatically?	YES
Breakable?		NO

OPERATION

Temporarily suspends security provisions for file at all (file, group, and account) levels, allowing any user in system unlimited access to the file. Suspension remains in effect until you enter a :SECURE command for this file to restore its previous security provisions. Suspension remains valid after session/job termination, or system failure followed by cold-load or reload. (However, it can be negated if system is re-loaded from a :SYSDUMP tape created before :RELEASE command was entered.)

NOTE

You can only use this command for a permanent disc file whose label identifies you as the creator of that file. When the normal (default) MPE security provisions are in effect, this must be a file in your log-on account and belonging to your log-on or home group.

This command does *not* affect the file's *lockword*, if any.

EXAMPLE

To release all security provisions for the file named FDATA, enter:

```
:RELEASE FDATA
```

TEXT DISCUSSION

Pages 6-61, 6-80.

:RENAME

Changes identity (file name, lockword, and/or group name) of disc file.

SYNTAX

```
:RENAME oldfilereference, newfilereference [,TEMP]
```

PARAMETERS

oldfilereference Current name of file, in following format:

filename [/lockword] [.groupname] [.accountname]

If account name is specified, it must be that of your log-on account. (Required parameter.)

newfilereference New name of file, in same format as *oldfilereference*. If account name is specified, it must be that of your log-on account. If group name is used, it must be one to which you have SAVE access. If account and/or group names are omitted, the log-on account and/or group are assumed. (Required parameter.)

TEMP

Indicates that old file was, and new file will be, temporary file in session/job temporary file domain. If omitted, permanent file is assumed. (Optional parameter.)

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	YES
	Programmatically?	YES
Breakable?		NO

OPERATION

Changes system file identification for permanent or temporary disc file. Effectively removes file with old name from system and creates another file with identical contents and new name. This command can be used to change the name of a file, to move a file from one group to another (by specifying a different group name in the *newfilereference* parameter), or to change the lockword.

NOTE

You can only apply the :RENAME command to a disc file that you yourself have created.

EXAMPLE

To change the name of a temporary file from OLDFILE to NEWFILE, moving it from your log-on group to the group named ORGB, enter:

```
:RENAME OLDFILE, NEWFILE. ORGB, TEMP
```

To change the lockword of the permanent file XFILE from LCKA to LCKB, enter:

```
:RENAME XFILE/LCKA, XFILE/LCKB
```

TEXT DISCUSSION Pages 6-21, 6-60, 6-61, 6-80, 6-81.

:REPORT

Displays accounting information for log-on account and group.

SYNTAX

```
:REPORT [,listfile]
```

PARAMETERS

listfile

Name of output file to which report is written. Automatically specified as new ASCII file with variable-length records, user-supplied carriage-control characters (CCTL), OUT access mode, and EXC (exclusive access) option. All other characteristics are same as :FILE command default specifications. If omitted, \$STDLIST is assigned by default.

NOTE

This parameter, if used, is always preceded by a delimiting comma.

(Optional parameter.)

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	YES
	Programmatically?	YES
Breakable?		YES (Aborts)

OPERATION

Outputs total accounting information logged against your log-on account and group. (If you have the Account Manager Capability, you receive information covering all groups under your log-on account.) The output includes usage counts and limits for permanent file space (in sectors), central processor time (in seconds), and session connect-time (in minutes). The file-space usage count reflects file space used as of the present moment, but the central processor time and connect-time usage counts reflect these counts as they were immediately prior to the start of the current session/job.

NOTE

If you have the System Manager or Account Manager Capability, you can issue a more explicit form of this command, referencing particular accounts and groups. Additionally, with the System Manager Capability, you can re-set resource use counts for an account and all of its groups. These functions are described in *System Manager/System Supervisor Manual*.

EXAMPLE

To obtain accounting information for your log-on account and group, enter the :REPORT command; the accounting information follows the listing of this command:

```
:REPORT
ACCOUNT          FILESPACE-SECTORS    CPU-SECONDS    CONNECT-MINUTES
  /GROUP          COUNT      LIMIT    COUNT      LIMIT    COUNT      LIMIT
TECHPUBS         128        2000        24         **        99         **
  /PUB           128        2000        24         **        99         **
```

TEXT DISCUSSION Pages 3-14, 5-6 through 5-8, 6-42.

:RESET

Re-sets formal file designator to original meaning.

SYNTAX

:RESET { <i>formaldesignator</i> } @

PARAMETERS

formaldesignator Formal file designator to be re-set to original meaning defined by program that references this designator. Contains from 1 to 8 alphanumeric characters, beginning with a letter. (Required parameter if @ not specified.)

@ Directive to re-set all formal file designators noted in all :FILE commands previously encountered in this session/job. (Required parameter if *formaldesignator* not specified.)

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	YES
	Programmatically?	YES
Breakable?		NO

OPERATION

Re-sets the formal designator for a file to its original meaning, negating the effect of any :FILE command that references this formal designator encountered up to this point. Applies to files on any device.

EXAMPLE

To cancel the effects of a previous :FILE command that specified characteristics for a file programmatically referred to as ALPHA, enter:

```
:RESET ALPHA
```

TEXT DISCUSSION Page 6-54.

:RESETDUMP

Disables Stackdump Facility.

SYNTAX

:RESETDUMP

PARAMETERS

None

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	YES
	Programmatically?	YES
Breakable?		NO

OPERATION

Disarms MPE Stackdump Facility (armed by :SETDUMP command), described in *MPE Debug/Stackdump Reference Manual*. When entered in BREAK, does *not* modify state of processes already created. If Stackdump Facility is not enabled, :RESETDUMP command has no effect.

EXAMPLE

To disable the Stackdump Facility, enter:

```
:RESETDUMP
```

TEXT DISCUSSION

MPE Debug/Stackdump Reference Manual.

:RESTORE

Returns to system one or more files stored off-line on magnetic tape by :STORE or :SYSDUMP command.

SYNTAX

```

:RESTORE tapefile [ [ file [ .group [ .account ] ] [ [ [ @ [ .group [ .account ] ] [ [ @ [ .@ [ .account ] ] ] ] ] ] ] ]
[;KEEP] [;DEV=device] [;SHOW] [;FILES=maxfiles] ]

```

PARAMETERS

<i>tapefile</i>	Name of magnetic tape file on which files to be retrieved now reside. This file must be referenced in the back-reference (*) format; this format references a previous :FILE command that defines the file as a magnetic tape file. A message is output to the Console Operator requesting him to mount the tape identified by the <i>filereference</i> parameter in the :FILE command, and allocate the tape unit to you. (Required parameter.)
<i>file.group.account</i>	Return file named in group and account designated.
<i>file.group</i>	Return file named in group designated under log-on account.
<i>file</i>	Return file named under log-on group.
<i>@.group.account</i>	Return all files in group named under account designated.
<i>@.group</i>	Return all files in group named under log-on account.
<i>@</i>	Return all files in log-on group.
<i>@.@.account</i>	Return all files in all groups under account named.
<i>@.@</i>	Return all files in all groups under log-on account.
<i>@.@.@</i>	Return all files in system.

NOTE

Sets of parameters in the format

$$\left[\begin{array}{l} \textit{file} [\textit{.group} [\textit{.account}]] \\ \textit{@} \left[\begin{array}{l} \textit{.group} [\textit{.account}]] \\ \textit{@} \left[\begin{array}{l} \textit{.@} \\ \textit{.account} \end{array} \right] \end{array} \right] \end{array} \right]$$

are referred to collectively as a *fileset*, and denote the files to be restored from tape. (Each file is restored as a permanent file.) The number of filesets specified is limited as follows: up to 10 by account name; up to 15 by account name and group name; up to 20 by account name, group name, and file name. If you specify no file set, the default assigned is @.@.@ (all files on the tape).

(Optional parameters.)

:RESTORE

(Continued)

- KEEP** Specification that if a file referenced in the :RESTORE command currently exists on disc, the file on disc is kept and the corresponding tape file is not copied into the system. If KEEP is omitted, and an identically-named file exists in the system, that file is replaced with the one on the tape. If KEEP is omitted, *and* a file on tape is eligible for restoring *and* a file of the same name exists on disc, *and* this disc file is busy, the disc file is kept and the tape file is not restored. (Optional parameter.)
- device* Device class name of devices or logical number of device on which all files are to be restored. (This name is also written on the label of each file restored.) If you omit this parameter, MPE attempts to replace the files on a device of the same class (or logical device number) as that of the device on which the file was created. If this attempt fails, perhaps because the device class specified does not exist or the tape was created on a previous version of this computer, MPE attempts to replace each file on a disc of the same type (fixed or moving-head) and subtype as that on which it was created. If this fails, MPE attempts to restore the file to a device of class name DISC. If this fails, the file is not restored. By judiciously categorizing disc device classes at your installation, your System Supervisor can automatically ensure appropriate replacement. (Optional parameter.)
- SHOW** Request to list names of restored files. If you omit SHOW, only total number of files restored, list of files not restored (and the reason each was not restored), and count of files not restored, are listed. (Optional parameter.)
- maxfiles* Maximum number of files that may be restored. If omitted, 4000 is assigned by default. (Optional parameter.)

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	YES
	Programmatically?	YES
Breakable?	YES (Aborted.)	

OPERATION

Reads back into system, on disc, a file or files stored off-line by :STORE or :SYSDUMP commands. If you have System Manager or System Supervisor Capability, you can restore any file from a :STORE tape, assuming the account and group to which the file belongs and the user who created the file are defined in the system. If you have Account Manager Capability, you can restore any file in your account (but cannot restore those with negative file codes unless you also have Privileged Mode Capability). If you have standard user capability only, you can restore any tape file in your log-on account if you have SAVE access to the group to which the file belongs; you cannot, however, restore those with negative file codes unless you have Privileged Mode Capability. If the file on tape is protected by a lockword, you must supply the lockword in the :RESTORE command. (If you are logged-on interactively, you are prompted for omitted lockwords.) However, if you are a System Manager, System Supervisor, or Account Manager (restoring within your own account), you are not required to provide passwords.

:RESTORE

(Continued)

The listing output by :RESTORE is transmitted to a file whose formal file designator is SYSLIST; if you do not specify otherwise, this file is equated, by default, to the standard list device (\$STDLIST).

NOTE

Before issuing a :RESTORE command, you must identify *tapefile* as a magnetic tape file. Do this via the :FILE command, written in the following format, including no parameters other than those shown:

```
:FILE formaldesignator [= filereference];DEV=device
```

The *device* parameter must indicate the device class name or logical unit number of a magnetic tape unit. All other parameters for *tapefile* are supplied by the :RESTORE command executor; if you attempt to supply any of these yourself, MPE rejects the :RESTORE command.

EXAMPLE

To retrieve from the tape file named BACKUP all files belonging to your log-on group, enter:

```
:FILE T=BACKUP; DEV=TAPE  
:RESTORE *T; @; KEEP; DEV=MDISC; SHOW
```

In response, the Console Operator receives a request to mount the tape identified as *BACKUP*. When that request is satisfied, the restore operation begins. If a tape file satisfying the @ specification already exists in the system, it is not restored, since the KEEP option was specified.

TEXT DISCUSSION Pages 6-64, 6-66 through 6-74, 7-19.

:RESUME

Resumes execution of suspended program.

SYNTAX

:RESUME

PARAMETERS

None

USE

Available	In Session?	YES (While in BREAK mode only)
	In Job?	NO
	In Break?	YES
	Programmatically?	NO
Breakable?		NO

OPERATION

After you suspend a program or MPE command operation by pressing the BREAK key or by invoking the CAUSEBREAK intrinsic, the :RESUME command resumes execution of that program or operation at the point where suspension occurred.

NOTE

The :RESUME command is legitimate *only* during a *Break*. Many MPE commands are *aborted* rather than suspended by a Break, and thus cannot be resumed.

EXAMPLE

:RESUME

TEXT DISCUSSION

Pages 3-16 through 3-18.



SYNTAX

<code>:RJE [commandfile] [,inputfile] [,listfile] [,punchfile]]]</code>

PARAMETERS

commandfile Actual designator of file from which Emulator reads its directives. Can be an ASCII input file. (Emulator's formal designator for this file is RJE~~C~~OM.) If omitted, MPE assigns \$STDIN (session/job input device). (Optional parameter.)

inputfile Actual designator of file from which Emulator reads input data to be transmitted to the remote computer. Can be any ASCII input file. (Emulator's formal designator for this file is RJEIN.) If omitted, MPE assigns \$STDIN (session/job input device).

listfile Actual designator of file to receive listed output obtained from remote computer. Can be any ASCII output file. (Emulator's formal designator for this file is RJE~~L~~IST.) Since this file is typically a non-disc file, it is usually pre-defined in a :FILE command and back-referenced in this format:

[**listfile*]

If omitted, MPE assigns \$STDLIST (session/job listing device). (Optional parameter.)

punchfile Actual designator of file to receive punched output obtained from remote computer. Can be any output file. (Emulator's formal designator for this file is RJE~~P~~UNCH.) Since this file is typically a non-disc file, it is usually pre-defined in a :FILE command and back-referenced in this format:

[**punchfile*]

If omitted, MPE assigns \$OLDPASS (if it exists) or \$NEWPASS (if \$OLDPASS does not exist). (Optional parameter.)

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	NO
	Programmatically?	NO
Breakable?	YES (Suspended.)	

OPERATION

Invokes the HP 2780/3780 Emulator. This subsystem permits transfer of data between an HP 3000 Series II System and a variety of remote processors in a full multiprogramming environment. The Emulator makes the HP 3000 computer appear to the remote processor as either an IBM 2780 or 3780 Data Transmission Terminal.

:RJE

(Continued)

EXAMPLE

To invoke the Emulator in session mode, enter directives to it via your standard input device, and input data via the disc file MYDATA, with listing output directed to a line printer file (LISTFL) and punched output sent to a card punch file (PUNCHFL), specify:

```
:FILE LISTFL; DEV=LP  
:FILE PUNCHFL; DEV=CP  
:RJE      , MYDATA, *LISTFL, *PUNCHFL
```



Note omitted parameter (for default file).

TEXT DISCUSSION *HP 2780/3780 Emulator Reference Manual.*

SYNTAX

`:RPG [textfile] [, [uslfile] [, [listfile] [, [masterfile] [, newfile]]]]`

PARAMETERS

textfile Actual designator of input file from which source program is read. Can be any ASCII input file. (Compiler's formal designator for this file is RPGTEXT.) If omitted, MPE assigns default designator \$STDIN (current input device). (Optional parameter.)

uslfile Actual designator of user subprogram library (USL) file on which object program is written. Can be any binary output file with file code of USL (or 1024). (Compiler's formal designator for this file is RPGUSL.) If you enter this parameter, it must indicate a file previously created in one of three ways:

1. By saving a USL file (through the :SAVE command) created by a previous compilation where the default value was used for the *uslfile* parameter.
2. By building the USL (through the MPE Segmenter command -BUILDUSL, discussed in *MPE Segmenter Reference Manual*).
3. By creating a new USL file (through the :BUILD command, with the mnemonic USL or decimal code of 1024 used for the *filecode* parameter).

If you omit *uslfile* parameter, MPE assigns as default either \$OLDPASS (if a passed file currently exists in session/job) or \$NEWPASS (if no passed file exists). (Optional parameter.)

listfile Actual designator of file on which program listing is written. This can be any ASCII output file. (Compiler's formal designator for this file is RPGLIST.) If omitted, MPE assigns default designator \$STDLIST (typically the terminal in a session or the printer in a batch job). (Optional parameter.)

masterfile Actual designator of file to be optionally merged with *textfile* and optionally written onto a file named *newfile*, as discussed in the manuals covering compilers. Can be any ASCII input file. (Compiler's formal designator for this file is RPGMAST.) If omitted, no merging takes place. (Optional parameter.)

newfile Actual designator of file on which (re-sequenced) records from *textfile* (and *masterfile*) are placed. Can be any ASCII output file. (Compiler's formal designator for this file is RPGNEW.) If omitted, no text is written to *newfile*. (Optional parameter.)

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	NO
	Programmatically?	NO
Breakable?	YES (Suspended.)	

:RPG

(Continued)

OPERATION

Compiles source-language program written in RPG onto a USL file on disc. If you do not specify a source text file, MPE expects input from your standard session/job input device. If you create the USL prior to compilation, you must assign it a *filecode* of USL (or 1024). If you do not specify a list file, MPE transmits the program listing to your standard session/job listing device. On the USL, each program unit exists as a relocatable binary module (RBM) that contains object code plus header information that labels and describes this code. You can compile RBMs onto the same USL during several compilations.

EXAMPLE

To compile an RPG program that you enter from your session/job input device into an object program in the USL file \$NEWPASS, and write the listing to your standard listing device, enter the following command. (If the next command is one to prepare an object program, \$NEWPASS can be passed to that command as an input file named \$OLDPASS.)

```
:RPG
```

To compile an RPG program residing on the disc file SOURCE into an object program on the USL file OBJECT, with a program listing generated on the disc file LISTFL, enter:

```
:BUILD OBJECT; CODE=USL
```

Creates USL File OBJECT in special format.

```
:RPG SOURCE, OBJECT, LISTFL
```

Compiles program from SOURCE onto OBJECT, creating LISTFL and writing listing to it.

TEXT DISCUSSION Page 7-6.

:RPGGO

Compiles, prepares, and executes an RPG program.

SYNTAX

```
:RPGGO [textfile] [, [listfile] [, [masterfile] [, [newfile]]]
```

PARAMETERS

<i>textfile</i>	Actual designator of input file from which source program is read. Can be any ASCII input file. (Compiler's formal designator for this file is RPGTEXT.) If omitted, MPE assigns default designator \$STDIN (current input device). (Optional parameter.)
<i>listfile</i>	Actual designator of file on which program listing is written. This can be any ASCII output file. (Compiler's formal designator for this file is RPGLIST.) If omitted, MPE assigns default designator \$STDLIST (typically the terminal in a session or the printer in a batch job). (Optional parameter.)
<i>masterfile</i>	Actual designator of file to be optionally merged with <i>textfile</i> and optionally written onto a file named <i>newfile</i> , as discussed in the manuals covering compilers. Can be any ASCII input file. (Compiler's formal designator for this file is RPGMAST.) If omitted, no merging takes place. (Optional parameter.)
<i>newfile</i>	Actual designator of file on which (re-sequenced) records from <i>textfile</i> (and <i>masterfile</i>) are placed. Can be any ASCII output file. (Compiler's formal designator for this file is RPGNEW.) If omitted, no text is written to <i>newfile</i> . (Optional parameter.)

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	NO
	Programmatically?	NO
Breakable?	YES (Suspended.)	

OPERATION

Compiles, prepares, and allocates/executes an RPG program. If you do not specify a source file, MPE expects your input from your standard session/job input device. If you do not specify a listing file, MPE writes your listing to your standard session/job listing device. This command creates a temporary user subprogram library (USL) file (\$NEWPASS) that you *cannot* access, and a temporary program file that you *can* access under the name \$OLDPASS.

EXAMPLE

To compile, prepare, and execute an RPG program entered from your standard input device, with the program listing transmitted to your standard listing device, enter:

```
:RPGGO
```

To compile, prepare, and execute an RPG program read from the disc file SOURCE and transmit the resulting program listing to the disc file LISTFL, enter:

```
:RPGGO SOURCE, LISTFL
```

TEXT DISCUSSION

Page 7-8.

:RPGPREP

Compiles and prepares an RPG program.

SYNTAX

```
:RPGPREP [ textfile ] [, [progfile] [, [listfile] [, [masterfile] [, [newfile]]]]
```

PARAMETERS

textfile Actual designator of input file from which source program is read. Can be any ASCII input file. (Compiler's formal designator for this file is RPGTEXT.) If omitted, MPE assigns default designator \$STDIN (current input device). (Optional parameter.)

progfile Actual designator of program file on which prepared program segments are written. Can be any binary output file with file code of PROG (or 1029). If you enter this parameter, it must indicate a file created in one of two ways:

1. By creating a new program file (through the :BUILD command, with the mnemonic PROG or decimal code of 1029 used for the *filecode* parameter).

NOTE

A program file is limited to only one disc extent. Thus, the *numextents* parameter of the :BUILD command must be one.

2. By specifying a non-existent file in the *progfile* parameter, in which case a session/job temporary file of the correct size and type is created.

If omitted, the default file \$NEWPASS is assigned. (Optional parameter.)

listfile Actual designator of file on which program listing is written. This can be any ASCII output file. (Compiler's formal designator for this file is RPGLIST.) If omitted, MPE assigns default designator \$STDLIST (typically the terminal in a session or the printer in a batch job). (Optional parameter.)

masterfile Actual designator of file to be optionally merged with *textfile* and optionally written onto a file named *newfile*, as discussed in the manuals covering compilers. Can be any ASCII input file. (Compiler's formal designator for this file is RPGMAST.) If omitted, no merging takes place. (Optional parameter.)

newfile Actual designator of file on which (re-sequenced) records from *textfile* (and *masterfile*) are placed. Can be any ASCII output file. (Compiler's formal designator for this file is RPGNEW.) If omitted, no text is written to *newfile*. (Optional parameter.)

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	NO
	Programmatically?	NO
Breakable?		YES (Suspended.)

:RPGPREP

(Continued)

OPERATION

Compiles and prepares an RPG program onto a program file on disc. If you do not specify a source file, MPE expects your input from your standard session/job input device. If you create the program file prior to compilation, you must assign it a *filecode* of PROG (or 1029). If you do not specify a list file, MPE transmits the listing output to your session/job listing device. The user subprogram library (USL) file created during compilation is a temporary file passed directly to the preparation mechanism; you can access it under the name \$OLDPASS *only if* the program file is not \$NEWPASS. The program file is also opened as a temporary file. For both files, the Segmenter searches first for a file of the proper name in the session/job temporary file domain; if such a file cannot be found, it then searches for a permanent file of that name. If no program file of the referenced name exists, the Segmenter creates a new program file that is saved in the session/job temporary file domain and prepares into this.

EXAMPLES

To compile and prepare an RPG program entered through the session/job input device, onto the file \$NEWPASS, with the listing printed on the session/job listing device, enter the following command. (If the next command is one to execute the program, the file \$NEWPASS is referenced in the execute command under the name \$OLDPASS.)

```
:RPGPREP
```

To compile and prepare an RPG source program input from a source file named SFILE into a program file named MYPROG, with the resulting listing generated on the session/job listing device, enter the following command:

```
:RPGPREP SFILE, MYPROG
```

TEXT DISCUSSION Pages 7-6, 7-7.

SYNTAX

```
:RUN progfile[,entrypoint]  
  [;NOPRIV]  
  [;LMAP]  
  [;DEBUG]  
  [;MAXDATA=segsz]  
  [;PARM=parameternum]  
  [;STACK=stacksize]  
  [;DL=dlsize]  
  
  [;LIB = {  $\begin{matrix} G \\ P \\ S \end{matrix}$  } ]  
  
  [;NOCB]
```

PARAMETERS

<i>progfile</i>	Actual designator of program file that contains prepared program. (Required parameter.)
<i>entrypoint</i>	Program entry-point where execution is to begin. May be primary entry point of program, or any secondary entry point in program's outer block. If parameter is omitted, primary entry point is assigned by default. (Optional parameter.)
NOPRIV	Declaration that program segments will be placed in <i>non-privileged</i> (user) mode. This parameter is intended for programs prepared with privileged-mode capability. Normally, program segments containing privileged instructions are executed (run) in privileged mode <i>only</i> if program was <i>prepared</i> with privileged-mode (PM) capability-class. (A program containing legally-compiled privileged code, placed in non-privileged mode, may abort when an attempt is made to execute it.) If NOPRIV is specified in :RUN command, all program segments are placed in <i>non-privileged</i> mode. (Library segments are not affected since their mode is determined independently.)
NOTE	
NOPRIV produces the same effect as omitting the \$OPTION PRIVILEGED and OPTION PRIVILEGED entries in your SPL source input.	
(Optional parameter.)	
LMAP	Request to produce a descriptive listing of the allocated (loaded) program on file whose formal designator is LOADLIST; if no :FILE command referencing LOADLIST is encountered, listing is produced on session/job listing device. If LMAP is omitted, listing is not produced. (Optional parameter.)
DEBUG	A request to insert a Debug call before the first executable instruction of the program. This parameter is ignored when non-privileged user runs a program having privileged mode capability. The parameter is also ignored if user does not have read and write access to program file. If omitted, no breakpoint is set. (Optional parameter.)

<i>segsiz</i>	Maximum stack area (Z-DL) size permitted, <i>in words</i> . This parameter is included if you expect to change size of DL-DB or Z-DB areas during program execution. If omitted, MPE assumes that you will not change these areas. (Optional parameter.)
<i>parameternum</i>	Value that can be passed to your program as a general parameter for control or other purposes; when program is executed, this value can be retrieved from address Q(initial)-4 where Q(initial) is Q-address for outer block of program. Value can be octal number or signed or unsigned decimal number. If <i>parameternum</i> is omitted, Q(initial)-4 address is filled with zeros. (Optional parameter.)
<i>stacksize</i>	Size of your initial local data area (Z-Q(initial)) in stack, in <i>words</i> . (This value <i>must</i> exceed 511 words.) This overrides <i>stacksize</i> estimated by Segmenter, which applies if <i>stacksize</i> parameter is omitted. (Default is a function of estimated stack requirements for each program unit in program. Since it is difficult for system to predict behavior of stack at run time, you may want to override default by supplying your own estimate with <i>stacksize</i> .) (Optional parameter.)
<i>dlsiz</i>	DL-DB area to be initially assigned to stack. This area is of interest mainly in programmatic applications. In all cases, the DL-DB area is rounded upward so that the distance from the beginning of the stack data segment to the DB-address is a multiple of 128 words. If <i>dlsiz</i> parameter is omitted, a value estimated by Segmenter applies. (Optional parameter.)
G	Search segmented procedure libraries in the following order to satisfy external references during allocation: Group library, followed by account public library, followed by system library.
P	Search segmented procedure libraries in the following order to satisfy external references during allocation: Account public library followed by system library.
S	Search segmented procedure libraries in the following order to satisfy external references during allocation: System library only.

NOTE

If G, P, and S are all omitted, S is assumed. (Optional parameters.)

NOCB	Request that file system not use stack segment (PCBX) for its control blocks, even if sufficient space is available. This will permit you to expand your stack (via the DLSIZE or ZSIZE intrinsics) to the maximum possible limit at a later time, but will cause the File Management System to operate more slowly for this program.
------	---

NOTE

You should only use this parameter if program absolutely requires largest stack possible.

NOTE

For the *stacksize*, *dlsiz*e, and *segsiz*e parameters, a value of - 1 indicates that the Segmenter is to assign the default value; it is equivalent to omitting the parameter. If values for *segsiz*e, *stacksiz*e, and *dlsiz*e are explicitly specified in the :RUN command, they override such values assigned at preparation time (which are recorded in the program file). If any of these parameters are omitted, the corresponding values recorded in the program file take effect.

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	NO
	Programmatically?	NO
Breakable?		YES (Suspended.)

OPERATION

Executes a program prepared in a program file. This command permits you to specify searching of segmented libraries (SLs), but not relocatable libraries (RLs), to satisfy external references.

EXAMPLE

To run a program stored in the program file PROG3, with no special parameters specified and all default values assumed, enter:

```
:RUN PROG3
```

To run a program stored in the program file PROG4, beginning at the entry-point SECLAB, with all segments running in non-privileged mode, enter:

```
:RUN PROG4, SECLAB; NOPRIV
```

TEXT DISCUSSION Pages 7-4, 7-9, 7-12, 7-19.

:SAVE

Saves file in system file domain.

SYNTAX

<code>.SAVE { \$OLDPASS,<i>newfilereference</i> } { <i>tempfilereference</i> }</code>

PARAMETERS

\$OLDPASS File currently being passed. After this file is saved, no file in your session/job temporary file domain can be referenced by the name \$OLDPASS. (Required parameter if *newfilereference* is also used.)

newfilereference New actual file designator to be assigned \$OLDPASS when it is made permanent, written in format:

filename[/lockword] [.groupname] [.accountname]

If *accountname* is used, this must indicate the log-on account. If *groupname* is used, this must indicate a group to which you have SAVE access, as defined by your Account Manager. If *groupname* is omitted, log-on group is assigned. (Required parameter if \$OLDPASS is used.)

tempfilereference Actual designator of temporary file to be made permanent under the same designator. The file is deleted from session/job temporary file domain and entered in system file domain. This parameter is written in following format:

filename[/lockword] [.groupname] [.accountname]

If *accountname* is used, this must indicate the log-on account. If *groupname* is used, this must indicate a group to which you have SAVE access, as defined by your Account Manager. If *groupname* is omitted, log-on group is assigned. (Required parameter if \$OLDPASS/*newfilereference* is not used.)

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	YES
	Programmatically?	YES
Breakable?		NO

OPERATION

Saves temporary file presently in session/job temporary file domain by converting it to permanent file in system file domain. Frequently used to save temporary \$OLDPASS files produced by compilation or preparation.

NOTE

This command applies to temporary files on disc only.

:SAVE

(Continued)

EXAMPLE

To save the temporary file \$OLDPASS, containing an object program produced by the FORTRAN compiler, enter the following :SAVE command. This retains the file in the system under the name OBFIL.

```
:FORTRAN TEXTFL
```

Compiles source text from TEXTFL onto file \$NEWPASS.

```
→ :SAVE $OLDPASS, OBFIL
```

Saves object program (written to \$NEWPASS, now referenced as \$OLDPASS) under file name OBFIL.

To save the temporary disc file TEMPFIL as a permanent file of the same name, enter:

```
:SAVE TEMPFIL
```

TEXT DISCUSSION Pages 3-4, 4-2, 4-3, 6-11, 6-63, 6-64, 7-4, 7-6, 7-8, 7-9.

:SECURE

Restores security provisions for a file that were suspended by :RELEASE command.

SYNTAX

`:SECURE filereference`

PARAMETERS

filereference


Actual designator of the disc file whose security provisions are to be restored, written in this format:

filename[*/lockword*] [*.groupname*] [*.accountname*]

If the file has a lockword, you must specify it. (Require parameter.)

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	YES
	Programmatically?	YES
Breakable?		NO



OPERATION

Restores all security provisions for a file that were previously suspended by :RELEASE command in this or another session/job.

NOTE

You can only use this command for a permanent disc file whose label identifies you as the creator of that file. When the normal (default) MPE security provisions are in effect, this must be a file in your log-on account and belonging to your log-on or home group.

EXAMPLE

To restore the security provisions previously in effect for the file named FDATA, enter:

```
:SECURE FDATA
```

TEXT DISCUSSION

Pages 6-62, 6-81.

:SEGMENTER

Calls MPE Segmenter.

SYNTAX

```
:SEGMENTER [listfile]
```

PARAMETERS

listfile

Actual designator of file to receive listable output from Segmenter. Can be any ASCII output file. (Segmenter's formal designator for this file is SEGLIST.) Since this file is typically a non-disc file, it is usually pre-defined in a :FILE command and back-referenced in this format:

[**listfile*]

If omitted, MPE assigns \$STDLIST (session/job listing device). (Optional parameter.)

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	NO
	Programmatically?	NO
Breakable?		YES (Suspended.)

OPERATION

Invokes MPE Segmenter Subsystem, used to create, delete, activate, and de-activate relocatable binary modules (RBMs) within a user subprogram library (USL), and to manage procedure libraries used to resolve external program references.

EXAMPLE

To invoke the Segmenter from a session and transmit the listable output to a line printer (rather than your terminal) enter:

```
:FILE LISTSEG; DEV=LP  
:SEGMENTER *LISTSEG
```

TEXT DISCUSSION

MPE Segmenter Reference Manual.

:SETDUMP

Enables Stack Dump on abort.

SYNTAX

```
:SETDUMP [DB[,ST[,QS]] [;ASCII]]
```

PARAMETERS

DB	Dump memory from DL to Q(initial) address. (Optional parameter.)
ST	Dump memory from Q(initial) to S address. (Optional parameter.)
QS	Dump memory from Q-63 to S address. (This parameter is ignored if ST also appears in parameter list.) (Optional parameter.)

NOTE

If you omit DB, ST, and QS, display shows settings of all registers at time of abort, and stack-marker trace.

ASCII	Dump ASCII conversion of octal display requested by above parameters, along with this display. (Optional parameter.)
-------	--

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	YES
	Programmatically?	YES
Breakable?		NO

OPERATION

Arms Stackdump Facility described in *MPE Debug/Stackdump Reference Manual*. This facility, when armed for a running program (process), takes effect if that process terminates abnormally. If the process belongs to a batch job, then the stack is dumped (in whole or in part) on the standard listing device just before the process terminates. If the process belongs to an interactive session, before the process terminates, MPE calls DEBUG to allow you to interactively analyze the stack contents and enabling you, upon return from DEBUG, to continue in sequence or exit back to your main line of code in attempt to recover. The :SETDUMP command arms the main process as well as all processes subsequently created under it (via commands such as :RUN, :SPL, :FORTPREP, :COBOLGO, and so forth). Arming remains in effect until you enter the :RESETDUMP command. When :SETDUMP is entered during a BREAK, it does *not* modify the state of processes already created. The Stackdump Facility is only armed between matching pairs of :SETDUMP/:RESETDUMP commands in session/job input stream. Two :SETDUMP commands without an intervening :RESETDUMP modify the setting mechanism according to the latest :SETDUMP command, except that in BREAK mode, the command does not take effect until another process is initiated from the Command Interpreter.

NOTE

In an interactive session, all :SETDUMP command parameters are ignored and the only effect of the command is to arm the Stackdump Facility in order for the process to call DEBUG if an abort occurs.

:SETDUMP

(Continued)

EXAMPLE

To arm the Stackdump Facility to display the memory area from the Q(initial) to S address, and accompanying ASCII conversion of the octal data, enter:

```
:SETDUMP ST; ASCII
```

TEXT DISCUSSION *MPE Debug/Stackdump Reference Manual.*

:SETMSG

Disables or re-enables receipt of user or operator messages at standard list device.

SYNTAX

:SETMSG	{ OFF }
	{ ON }

PARAMETERS

OFF Sets session or job to refuse :TELL command messages from other users (and =TELL command messages from Console Operator).

ON Re-enables message reception.

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	YES
	Programmatically?	YES
Breakable?		NO

OPERATION

Allows you to run a session or job in *quiet mode*, so that :TELL messages from other users and =TELL messages from the Console Operator are refused. (However, =WARN messages from the Console Operator override quiet mode and are accepted.)

EXAMPLES

To prevent messages, enter:

```
:SETMSG OFF
```

To re-enable messages, enter:

```
:SETMSG ON
```

TEXT DISCUSSION Pages 9-3, 9-4.

:SHOWDEV

Reports status of input/output devices.

SYNTAX

<code>:SHOWDEV [<i>ldev</i> [<i>classname</i>]</code>
--

PARAMETERS

ldev Logical device number of device for which status information is to be displayed. Unique for each individual device. (Optional parameter.)

classname Device class name of device(s) for which status information is to be displayed. May apply to several devices. (Optional parameter.)

NOTE:

If both *ldev* and *classname* are omitted, status information for all devices on the system is displayed.

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	YES
	Programmatically?	YES
Breakable?	YES (Aborted.)	

OPERATION

Displays status information about any input/output device on the system. The following items may appear in the listing, always transmitted to your standard session/job listing device:

- Logical device number.
- Input type allowed:
 - J = Accepts sessions/jobs.
 - D = Accepts data.
 - A = Accepts sessions/jobs *and* data.
- Availability:
 - AVAIL = Available as a real, non-sharable device.
 - SPOOLED = Available via input or output spooling.
 - UNAVAIL = Unavailable (owned by a session/job).
 - DISC = Device is a disc, always available.
- Device ownership (if applicable):
 - SYS = Owned by system; if #*nnn* appears, it specifies the Process Identification Number (PIN) of the owning process (running program).
 - SPOOLER IN = Input spooling in effect, owned by spooler.
 - SPOOLER OUT = Output spooling in effect, owned by spooler.

:SHOWDEV

(Continued)

#Jnnn	=	Owned by indicated job.
#Snnn	=	Owned by indicated session.
DIAG	=	Allocated to diagnostic testing by Console Operator via =GIVE command.
nnn FILES	=	Indicates number of extents currently in use on a disc.
DOWN	=	Device is off-line (requested by Console Operator via =DOWN Console Command).
DP	=	Device is being taken off-line (=DOWN command operation pending).

The display appears in the following format:

LDEV	AVAIL	OWNERSHIP
20	A AVAIL	
21	A AVAIL	
22	A UNAVAIL	#S11: 2 FILES
23	A UNAVAIL	#S8: 2 FILES
24	A AVAIL	
25	A AVAIL	
26	A AVAIL	
27	A UNAVAIL	#S13: 4 FILES
28	A AVAIL	
29	A AVAIL	
30	A AVAIL	

EXAMPLES

To display the status of the device identified by Logical Device No. 12, enter:

```
:SHOWDEV 12
```

To report the status of all devices assigned the device class name LP, enter:

```
:SHOWDEV LP
```

To print the status of all input/output devices on the system, enter:

```
:SHOWDEV
```

TEXT DISCUSSION Pages 5-6, 8-1 through 8-3.

:SHOWIN

Reports status of input devicefiles.

SYNTAX

```
:SHOWIN [ #Innn  
          STATUS  
          SP  
          item[;item[;item]] ]
```

PARAMETERS

<i>#Innn</i>	Identifier of particular input devicefile for which information is to be displayed. The information appears in Type I Format, described below. (Optional parameter.)
STATUS	Request to summarize status information for all current input devicefiles. Type II Format. (Optional parameter.)
SP	Request to display status of all currently spooled input devicefiles. Type III Format. (Optional parameter.)
<i>item</i>	Request to display status of all current input devicefiles that satisfy the following qualifications:

[DEV = *ldev*]

$$\left[\text{JOB} = \left\{ \begin{array}{l} @S \\ @J \\ \#Snnn \\ \#Jnnn \end{array} \right\} \right]$$

[ACTIVE
 READY
 OPENED]

Each set of brackets, above, defines an *item*. You cannot use the same *item* more than once in the parameter list. The subparameters are:

ldev = Display status of input devicefile residing on device identified by logical device number *ldev*.

@S = Display status of input devicefiles for all sessions.

@J = Display status of input devicefiles for all jobs.

#Snnn = Display status of all input devicefiles for session indicated.

#Jnnn = Display status of all input devicefiles for job indicated.

ACTIVE
READY
OPENED } = Display status of input devicefiles in this state.

:SHOWIN

(Continued)

When information for only one devicefile is displayed, output appears in Type I Format; when information for more than one devicefile is displayed, output appears in Type III Format. (Optional parameters.)

NOTE

If you omit all parameters for this command, MPE displays status information for all input devicefiles used by your session/job.

Do *not* use duplicate *item* keywords in this command; that is, you can specify :SHOWIN DEV= 25; ACTIVE;@J but *not* :SHOWIN DEV= 25; ACTIVE; OPENED.

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	YES
	Programmatically?	YES
Breakable?	YES (Aborted.)	

OPERATION

Displays status information about one or more currently-defined input devicefiles. This information reflects the status at the time you enter the command, and always appears on your standard session/job listing device. The information includes:

- Logical device number of device.
- Devicefile identification in form #Innn.
- Session/job number (*sjnum*) of session/job using the devicefile, if not used for READY or ACTIVE data; otherwise, session/job name appears on line following standard device information.
- Filename associated with the devicefile.
- State:
 - ACTIVE = Input being read from spooled device to disc.
 - READY = Input spooling completed, and file is now ready for use by a program.
 - OPENED = File is being accessed by a program.
- Approximate disc space currently used (in sectors), for spooled input devicefiles only.
- Rank, indicating the order in which the file is entered in the system with respect to other files.
- Input Priority requested by user (1 = lowest, 13 = highest, blank = current default priority.)

Output may appear in three possible formats:

- Type I:

DEV/CL	DFID	JOBNUM	FNAME	STATE	FRM	SPACE	RANK	PRI	#C
45	#I25	#S16	\$STDIN	OPENED					

- Type II:

```
8 FILES:
  0 ACTIVE
  0 READY; INCL 0 SPOOFLES, 0 DEFERRED
  8 OPENED; INCL 0 SPOOFLES
  0 SPOOFLES: 0 SECTORS
```

- Type III:

(Type I followed by Type II.)

EXAMPLES

To display the status of the input devicefile with the identifier #I796, enter:

```
:SHOWIN #I796
```

To display the status of all spooled input devicefiles, enter:

```
:SHOWIN SP
```

To request summarized information about all input devicefiles, enter:

```
:SHOWIN STATUS
```

To report the status of all ACTIVE devicefiles for the session identified as #S432, enter:

```
:SHOWIN JOB=#S432; ACTIVE
```

TEXT DISCUSSION Pages 3-21, 5-6, 8-8 through 8-11.

:SHOWJOB

Displays status information
about sessions and jobs.



SYNTAX

<code>:SHOWJOB</code>	$\left[\begin{array}{l} \#Snnn \\ \#Jnnn \\ STATUS \\ id[;state] \\ state[;id] \end{array} \right]$
-----------------------	--

PARAMETERS

- #Snnn** The session number (assigned by MPE) of the session for which status information is to be displayed. The information appears in Type I Format, described below. (Optional parameter.)
- #Jnnn** The job number (assigned by MPE) of the job for which status information is to be displayed. Type I Format. (Optional parameter.)
- STATUS** A request to summarize information for all sessions/jobs. Type II Format. (Optional parameter.)
- id** A list of sessions/jobs whose status information is to be displayed, in this format:

$$\left[\text{JOB} = \left\{ \begin{array}{l} @S \\ @J \\ [sname,]username.acctname \\ @,username.acctname \\ [@,] @.acctname \end{array} \right\} \right]$$

The symbol @ indicates *all*; thus, @S means all sessions, @J means all jobs, and @,@.acctname means all sessions and jobs for all users in the account. Where only one session/job is displayed, output appears in Type I Format; where more than one is displayed, output appears in Type III Format. (Optional parameter.)

- state** A particular session/job state, specified as a further restriction on which sessions/jobs are to be displayed:

$$\left[\begin{array}{l} \text{INTRO} \\ \text{WAIT} \quad \left[\begin{array}{l} ,N \\ ,D \end{array} \right] \\ \text{EXEC} \end{array} \right]$$

N and D, which are requests for only non-deferred or deferred sessions/jobs respectively, apply only to the WAIT state.

Where only one session/job is displayed, output appears in Type I Format; where more than one is displayed, output appears in Type III Format. (Optional parameter.)

NOTE

If you omit the parameter list for the :SHOWJOB command, then all status information for all sessions/jobs in the system is displayed.

:SHOWJOB

(Continued)

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	YES
	Programmatically?	YES
Breakable?	YES (Aborted)	

OPERATION

Obtains status information about any currently-defined sessions or jobs.

Output may appear in three possible formats:

- Type I:

```
JOBNUM  STATE  IPRI  JIN  JLIST      INTRODUCED  JOB NAME
#S16    EXEC           45  45      THU 11:36A  DIX.MAC
JOBFENCE= 0; JLIMIT= 3; SLIMIT= 16
```

- Type II:

```
7 JOBS:
  0 INTRO
  0 WAIT; INCL 0 DEFERRED
  7 EXEC; INCL 7 SESSIONS
JOBFENCE= 0; JLIMIT= 3; SLIMIT= 16
```

- Type III:

(Type I followed by Type II.)

EXAMPLES

To display the status of all sessions, enter:

```
:SHOWJOB JOB=@S
```

To display the status of Session Number S16, enter:

```
:SHOWJOB #S16
```

To display summary information for all sessions and jobs, enter:

```
:SHOWJOB STATUS
```

To display the status of all jobs in the INTRO state, enter:

```
:SHOWJOB JOB=@J; INTRO
```

TEXT DISCUSSION Pages 3-10, 3-15, 4-12, 5-1 through 5-6, 9-1.

:SHOWOUT

Reports status of output devicefiles.

SYNTAX

```
:SHOWOUT [ #Onnn  
          STATUS  
          SP  
          item[;item[;item]] ]
```

PARAMETERS

<i>#Onnn</i>	Identifier of particular output devicefile for which information is to be displayed. The information appears in Type I Format, described below. (Optional parameter.)
STATUS	Request to summarize status information for all current output devicefiles. Type II Format. (Optional parameter.)
SP	Request to display status of all currently spooled output devicefiles. Type III Format. (Optional parameter.)
<i>item</i>	Request to display status of all current output devicefiles that satisfy the following qualifications:

$$\left[\text{DEV} = \left\{ \begin{array}{l} ldev \\ classname \end{array} \right\} \right]$$
$$\left[\text{JOB} = \left\{ \begin{array}{l} @S \\ @J \\ \#Snnn \\ \#Jnnn \end{array} \right\} \right]$$
$$\left[\begin{array}{l} \text{ACTIVE} \\ \text{READY} \left[\begin{array}{l} ,N \\ ,D \end{array} \right] \\ \text{OPENED} \\ \text{LOCKED} \end{array} \right]$$

Each set of brackets, above, defines an *item*. You cannot use the same *item* more than once in the parameter list. The subparameters are:

<i>ldev</i>	=	Display status of output devicefile residing on device identified by logical device number <i>ldev</i> .
<i>classname</i>	=	Display status of output devicefiles residing on all devices having device class name <i>classname</i> .
@S	=	Display status of output devicefiles for all sessions.
@J	=	Display status of output devicefiles for all jobs.
#Snnn	=	Display status of all output devicefiles for session indicated.
#Jnnn	=	Display status of all output devicefiles for job indicated.

:SHOWOUT

(Continued)

ACTIVE }
READY } = Display status of output
OPENED } devicefiles in this state.
LOCKED }

N = Display status of non-deferred READY devicefiles only.

D = Display status of deferred READY devicefiles only.

Where information for only one devicefile is displayed, output appears in Type I Format; where information for more than one devicefile is displayed, output appears in Type III Format. (Optional parameters.)

NOTE

If you omit all parameters for this command, MPE displays status information for *all* output devicefiles used by your session/job.

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	YES
	Programmatically?	YES
Breakable?		YES (Aborted.)

OPERATION

Displays status information about one or more currently-defined output devicefiles. This information reflects the status at the time you enter the command, and always appears on your standard session/job listing device. The information includes:

- Logical device number or device class name of device.
- Devicefile identification, in form #Onnn.
- Session/job number (*sjnum*) of session/job using devicefile, if not used for READY or ACTIVE data; otherwise, session/job name appears on line following standard device information.
- Filename assigned to devicefile.
- State

ACTIVE = Spooled devicefile on disc is actually being written to a printer, plotter, or card punch.

READY = Devicefile on disc is ready for output.

LOCKED = READY, but system is using file with exclusive access.

OPENED = Devicefile on disc is being accessed by a program.

:SHOWOUT

(Continued)

- Forms Message Indicator (the letter Y), appearing only if a forms alignment message applies to this devicefile.
- Approximate disc space currently used (in sectors), for spooled output devicefiles only.
- Rank, indicating the order in which the file is entered in the system with respect to other files.
- D, indicating a deferred file, for spooled devicefiles only.
- Outputpriority, requested by user, for spooled devicefiles only (1 = lowest, 13 = highest, blank = current default priority).
- Number of copies needed (#C), for spooled devicefiles only.

Output may appear in three possible formats:

- Type I:

```
DEV/CL  DFID    JOBNUM  FNAME    STATE FRM SPACE RANK PRI #C
45      #032    #S16    $STDLIST OPENED
```

- Type II:

```
7 FILES:
0 ACTIVE
0 READY; INCL 0 SPOOFLES, 0 DEFERRED
7 OPENED; INCL 0 SPOOFLES
0 SPOOFLES: 0 SECTORS
```

```
OUFENCE= 0
```

- Type III:

(Type I followed by Type II.)

EXAMPLES

To display the status of the output devicefile with the identifier #0977, enter:

```
:SHOWOUT #0977
```

To display the status of all spooled output devicefiles, enter:

```
:SHOWOUT SP
```

To request summarized information about all output devicefiles, enter:

```
:SHOWOUT STATUS
```


:SHOWOUT

(Continued)

To request the status of all READY non-deferred devicefiles for the session #S432, enter:

```
:SHOWOUT JOB=#S432;READY,N
```

TEXT DISCUSSION Pages 3-21, 4-19, 8-13 through 8-15.

:SHOWTIME

Prints current date and time.

SYNTAX

:SHOWTIME

PARAMETERS

None

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	YES
	Programmatically?	YES
Breakable?		NO

OPERATION

Prints current date and time, as indicated by system clock.

EXAMPLE

To display the date and time, enter:

```
:SHOWTIME  
THU, AUG 21, 1975, 5:16 PM
```

TEXT DISCUSSION

Page 9-12.

:SPEED

Changes terminal operating speed.

SYNTAX

```
:SPEED { [inspeed],outspeed }
        { inspeed }
```

PARAMETERS

inspeed New input speed, in characters per second. Must be 10, 14, 15, 30, 60, 120, or 240. (Required parameter when *outspeed* is omitted.)

outspeed New output speed, in characters per second. Must be 10, 14, 15, 30, 60, 120, or 240. (Optional parameter.)

USE

Available	In Session?	YES
	In Job?	NO
	In Break?	YES
	Programmatically?	YES
Breakable?		NO

OPERATION

MPE automatically senses the input/output speed of a terminal when you log on at that terminal. If your terminal has speed-adjustment controls, you can change the input and output speeds after log-on by entering the :SPEED command.

NOTE

This command is not valid for terminals that operate at only one speed.

When you enter the command, MPE outputs the following message (at the old output speed):

```
CHANGE SPEED AND INPUT "MPE"
```

In response, you then manually change the speed control on the terminal and verify the new speed by entering the message:

```
MPE
```

If the characters "MPE" cannot be verified, the system assumes that the terminal is to continue at the old speed.

EXAMPLES

To change the input speed to 240 cps and the output speed to 10 cps, enter:

```
:SPEED 240, 10
```

To change the input speed only to 120 cps, enter:

```
:SPEED 120
```


SYNTAX

```
:SPL [textfile] [, [uslfile] [, [listfile] [, [masterfile] [, [newfile]]]]
```

PARAMETERS

textfile Actual designator of input file from which source program is read. Can be any ASCII input file. (Compiler's formal designator for this file is SPLTEXT.) If omitted, MPE assigns default designator \$STDIN (current input device). (Optional parameter.)

uslfile Actual designator of user subprogram library (USL) file on which object program is written. Can be any binary output file with file code of USL (or 1024). (Compiler's formal designator for this file is SPLUSL.) If you enter this parameter, it must indicate a file previously created in one of three ways:

1. By saving a USL file (through the :SAVE command) created by a previous compilation where the default value was used for the *uslfile* parameter.
2. By building the USL (through the MPE Segmenter command -BUILDUSL, discussed in *MPE Segmenter Reference Manual*).
3. By creating a new USL file (through the :BUILD command, with the mnemonic USL or decimal code of 1024 used for the *filecode* parameter).

If you omit *uslfile* parameter, MPE assigns as default either \$OLDPASS (if a passed file currently exists in session/job) or \$NEWPASS (if no passed file exists). (Optional parameter.)

listfile Actual designator of file on which program listing is written. This can be any ASCII output file. (Compiler's formal designator for this file is SPLLIST.) If omitted, MPE assigns default designator \$STDLIST (typically the terminal in a session or the printer in a batch job). (Optional parameter.)

masterfile Actual designator of file to be optionally merged with *textfile* and optionally written onto a file named *newfile*, as discussed in the manuals covering compilers. Can be any ASCII input file. (Compiler's formal designator for this file is SPLMAST.) If omitted, no merging takes place. (Optional parameter.)

newfile Actual designator of file on which (re-sequenced) records from *textfile* (and *masterfile*) are placed. Can be any ASCII output file. (Compiler's formal designator for this file is SPLNEW.) If omitted, no text is written to *newfile*. (Optional parameter.)

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	NO
	Programmatically?	NO
Breakable?	YES (Suspended.)	

:SPL

(Continued)

OPERATION

Compiles source-language program written in SPL onto a USL file on disc. If you do not specify a source text file, MPE expects input from your standard session/job input device. If you create the USL prior to compilation, you must assign it a *filecode* of USL (or 1024). If you do not specify a list file, MPE transmits the program listing to your standard session/job listing device. In the USL, each program unit exists as a relocatable binary module (RBM) that contains object code plus header information that labels and describes this code. You can compile RBMs onto the same USL during several compilations.

EXAMPLE

To compile an SPL program that you enter from your session/job input device into an object program in the USL file \$NEWPASS, and write the listing to your standard listing device, enter the following command. (If the next command is one to prepare an object program, \$NEWPASS can be passed to that command as an input file named \$OLDPASS.)

```
:SPL
```

To compile an SPL program residing on the disc file SOURCE into an object program on the USL file OBJECT, with a program listing generated on the disc file LISTFL, enter:

```
:BUILD OBJECT; CODE=USL
```

Creates USL File OBJECT in special format.

```
:SPL SOURCE, OBJECT, LISTFL
```

Compiles program from SOURCE onto OBJECT, creating LISTFL and writing listing to it.

TEXT DISCUSSION Pages 3-16, 7-5.

:SPLGO

Compiles, prepares, and executes
an SPL program.

SYNTAX

```
:SPLGO [textfile] [, [listfile] [, [masterfile] [, newfile]]]
```

PARAMETERS

<i>textfile</i>	Actual designator of input file from which source program is read. Can be any ASCII input file. (Compiler's formal designator for this file is SPLTEXT.) If omitted, MPE assigns default designator \$STDIN (current input device). (Optional parameter.)
<i>listfile</i>	Actual designator of file on which program listing is written. This can be any ASCII output file. (Compiler's formal designator for this file is SPLLIST.) If omitted, MPE assigns default designator \$STDLIST (typically the terminal in a session or the printer in a batch job). (Optional parameter.)
<i>masterfile</i>	Actual designator of file to be optionally merged with <i>textfile</i> and optionally written onto a file named <i>newfile</i> , as discussed in the manuals covering compilers. Can be any ASCII input file. (Compiler's formal designator for this file is SPLMAST.) If omitted, no merging takes place. (Optional parameter.)
<i>newfile</i>	Actual designator of file on which (re-sequenced) records from <i>textfile</i> (and <i>masterfile</i>) are placed. Can be any ASCII output file. (Compiler's formal designator for this file is SPLNEW.) If omitted, no text is written to <i>newfile</i> . (Optional parameter.)

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	NO
	Programmatically?	NO
Breakable?		YES (Suspended.)

OPERATION

Compiles, prepares, and allocates/executes an SPL program. If you do not specify a source file, MPE expects your input from your standard session/job input device. If you do not specify a listing file, MPE writes your listing to your standard session/job listing device. This command creates a temporary user subprogram library (USL) file (\$NEW-PASS) that you *cannot* access, and a temporary program file that you *can* access under the name \$OLDPASS.

EXAMPLE

To compile, prepare, and execute an SPL program entered from your standard input device, with the program listing transmitted to your standard listing device, enter:

```
:SPLGO
```

To compile, prepare, and execute an SPL program read from the disc file SOURCE and transmit the resulting program listing to the disc file LISTFL, enter:

```
:SPLGO SOURCE, LISTFL
```


:SPLPREP

Compiles and prepares an SPL program



SYNTAX

```
:SPLPREP [textfile] [, [progfile] [, [listfile] [, [masterfile] [, [newfile]]]]
```

PARAMETERS

textfile Actual designator of input file from which source program is read. Can be any ASCII input file. (Compiler's formal designator for this file is SPLTEXT.) If omitted, MPE assigns default designator \$STDIN (current input device). (Optional parameter.)

progfile Actual designator of program file on which prepared program segments are written. Can be any binary output file with file code of PROG (or 1029). If you enter this parameter, it must indicate a file created in one of two ways:

1. By creating a new program file (through the :BUILD command, with the mnemonic PROG or decimal code of 1029 used for the *filecode* parameter).

NOTE

A program file is limited to only one disc extent. Thus, the *numextents* parameter of the :BUILD command must be 1.

2. By specifying a non-existent file in the *progfile* parameter, in which case a session/job temporary file of the correct size and type is created.

If omitted, the default file \$NEWPASS is assigned. (Optional parameter.)

listfile Actual designator of file on which program listing is written. This can be any ASCII output file. (Compiler's formal designator for this file is SPLLIST.) If omitted, MPE assigns default designator \$STDLIST (typically the terminal in a session or the printer in a batch job). (Optional parameter.)

masterfile Actual designator of file to be optionally merged with *textfile* and optionally written onto a file named *newfile*, as discussed in the manuals covering compilers. Can be any ASCII input file. (Compiler's formal designator for this file is SPLMAST.) If omitted, no merging takes place. (Optional parameter.)

newfile Actual designator of file on which (re-sequenced) records from *textfile* (and *masterfile*) are placed. Can be any ASCII output file. (Compiler's formal designator for this file is SPLNEW.) If omitted, no text is written to *newfile*. (Optional parameter.)

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	NO
	Programmatically?	NO
Breakable?	YES (Suspended.)	

:SPLPREP

(Continued)

OPERATION

Compiles and prepares an SPL program onto a program file on disc. If you do not specify a source file, MPE expects your input from your standard session/job input device. If you create the program file prior to compilation, you must assign it a *filecode* of PROG (or 1029). If you do not specify a list file, MPE transmits the listing output to your session/job listing device. The user subprogram library (USL) file created during compilation is a temporary file passed directly to the preparation mechanism; you can access it under the name \$OLDPASS *only if* the program file is not \$NEWPASS. The program file is also opened as a temporary file. For both files, the Segmenter searches first for a file of the proper name in the session/job temporary file domain; if such a file cannot be found, it then searches for a permanent file of that name. If no program file of the referenced name exists, the Segmenter creates a new program file that is saved in the session/job temporary file domain and prepares into this.

EXAMPLE

To compile and prepare an SPL program entered through the session/job input device, onto the file \$NEWPASS, with the listing printed on the session/job listing device, enter the following command. (If the next command is one to execute the program, the file \$NEWPASS is referenced in the execute command under the name \$OLDPASS.)

```
:SPLPREP
```

To compile and prepare an SPL source program input from a source file named SFILE into a program file named MYPROG, with the resulting listing generated on the session/job listing device, enter the following command:

```
:SPLPREP SFILE, MYPROG
```

TEXT DISCUSSION Page 7-6.

:STORE

Stores one or more disc files
onto magnetic tape.

SYNTAX

```
:STORE [ file [.group [.account] ] [ @ [ .group [.account] ] [ .@ [.account] ] [ , ... ] ] ] ; tapefile  
[;SHOW] [;FILES=maxfiles]
```

PARAMETERS

<i>file.group.account</i>	Store file named in group and account designated.
<i>file.group</i>	Store file named in group designated under log-on account.
<i>file</i>	Store file named under log-on group.
<i>@.group.account</i>	Store all files in group named under account designated.
<i>@.group</i>	Store all files in group named under log-on account.
<i>@</i>	Store all files in log-on group.
<i>@.@.account</i>	Store all files in all groups under account named.
<i>@.@</i>	Store all files in all groups under log-on account.
<i>@.@.@</i>	Store all files in system.

NOTE

Sets of parameters in the format

$$\left[\begin{array}{l} \textit{file} \left[\begin{array}{l} \textit{.group} \left[\begin{array}{l} \textit{.account} \end{array} \right] \end{array} \right] \\ @ \left[\begin{array}{l} \textit{.group} \left[\begin{array}{l} \textit{.account} \end{array} \right] \\ \textit{.@} \left[\begin{array}{l} \textit{.account} \end{array} \right] \end{array} \right] \end{array} \right]$$

are referred to collectively as a *fileset*, and denote the *files* to be dumped to tape. If you do not specify a *fileset*, the default value is @ (all files of the log-on group are copied). If any file belonging to a fileset requires a lockword, you must supply a lockword. If you fail to do so while in batch mode, the file is not stored. If you fail to supply it in interactive mode, you are prompted for the lockword. (Optional parameters.)

<i>tapefile</i>	Name of destination tape file onto which the stored files are written. This can be any magnetic tape file from the output set. This file must be referenced in the back-reference (*) format; this format references a previous :FILE command that identifies the file as a magnetic tape file. (Required parameter.)
SHOW	Request to list names of file stored. If SHOW is omitted, total number of files stored, names of files not stored, and number of files not stored are listed. (Optional parameter.)
<i>maxfiles</i>	Maximum number of files that may be stored. If omitted, 4000 is specified by default. (Optional parameter.)

:STORE

(Continued)

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	YES
	Programmatically?	YES
Breakable?		YES (Aborted.)

OPERATION

If you have System Manager or System Supervisor Capability, you can store any user file in the system. If you have Account Manager Capability, you can store any file in your account (but cannot dump those with negative file codes unless you also have Privileged Mode Capability). You may request storing of one file alone or various sets of files.

NOTE

Before issuing a :STORE command, you must identify *tapefile* as a magnetic tape file. Do this via the :FILE command, written in the following format, including no parameters other than those shown:

```
:FILE formaldesignator [= filereference] ;DEV=device
```

The *device* parameter must indicate the device class name or logical unit number of a magnetic tape unit. All other parameters for *tapefile* are supplied by the :STORE command executor; if you attempt to supply any of these yourself, MPE rejects the :STORE command

If you press the BREAK key during the store operation, the operation stops after storing the current file, and further output is suppressed.

EXAMPLE

To copy all files in the group GP4X in your log-on account, to a tape file named BACKUP, enter the following commands. In response, the operator receives a request to mount the tape identified as *BACKUP*. A listing of the files copied appears on the standard list device.

```
:FILE T=BACKUP; DEV=TAPE  
:STORE @.GP4X; *T; SHOW
```

Explicit or implicit repeated references are permitted among the files requested, but once a file is locked down (in use by the system), any subsequent reference to it in the :STORE command results in the message BUSY even though execution of the :STORE command continues.

Suppose the file identified as FN.GN.AN is a member of the fileset referenced by @ in the following command:

```
:STORE @, FN.GN.AN; *DUMPTAPE; SHOW
```

The command is executed successfully, but the *fileset#* and *msg* notations under FILES NOT STORED on the listing will show:

<i>filename</i>	<i>groupname</i>	<i>accountname</i>	<i>fileset#</i>	<i>msg</i>
FN	GN	AN	2	BUSY

This same file will also be noted under *FILES STORED*.

TEXT DISCUSSION Pages 3-6, 6-64 through 6-70, 7-19.

:STREAM

Spools batch jobs or data in session
or job mode.

SYNTAX

```
:STREAM [inputfile] [,character]
```

PARAMETERS

inputfile

File designator of ASCII input file from which jobs or data are spooled to disc. May indicate disc file containing commands or data entered using Editor, or file input from card reader, tape unit, or terminal. Records in this file cannot exceed 255 characters. All character positions within a record are significant.

NOTE

If creating the input file with the Editor, be sure to omit line numbers from the records in this file when you SAVE it; do this by using the UNN parameter with the Editor's KEEP command.

Default: standard input device. (Optional parameter.)

character

Character used in place of colon (:) to identify MPE commands within *inputfile*. When *inputfile* is entered on device configured to accept sessions/jobs or data input via the :DATA command, this character can be any ASCII special (non-alphanumeric) character except a colon. Default: MPE assumes you are using exclamation point (!) or colon (:) as the substitute character.

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	YES
	Programmatically?	YES
Breakable?	YES (Command and partially-streamed job aborted.)	

OPERATION

Spools batch jobs and data from *inputfile* to disc, and schedules each job independently. Most commonly used for initiating batch jobs within sessions.

NOTE

For this command to take effect, Console Operator must first enable streaming (with =STREAMS command).

When you enter :STREAM with the terminal as the default input device, whether during a session or job, MPE prompts you for input by displaying a greater-than (>) character; when you enter :STREAM for another input device, MPE does not print the prompt character. In any case, you terminate the *inputfile* by entering the :EOD command. At a terminal, this halts prompting for job input and returns control to your session/job.

Begin each job in the *inputfile* with the JOB command and terminate it with the EOJ command. Begin each data file with the DATA command and terminate this file with the EOD command. (Begin all commands with an appropriate *substitute* command identifier, as in !JOB or !DATA.) When it spools the *inputfile* to disc, MPE replaces the substitute command identifier with a colon so that command records are properly interpreted when executed.

:TELL

Sends a message to another session or job.

SYNTAX

<pre>:TELL { #Snnn #Jnnn [sjname,] username.acctname } ; [text]</pre>

PARAMETERS

#Snnn Session number assigned by MPE to the session that is to receive the message.

#Jnnn Job number assigned by MPE to the job that is to receive the message.

sjname }
username }
acctname } Names of the session/job and user to receive the message, and the name of the account under which they are running — the *session/job identity* entered in the :HELLO or :JOB command used to initiate the receiving session or job.

If several users are running under the same *[sjname,] username.acctname*, then MPE will arbitrarily select only one of these. In such a case, you should use the *#Snnn* or *#Jnnn* parameter with the :TELL command.

NOTE

You can obtain the identities and numbers for all current sessions/jobs as indicated in the :SHOWJOB command specifications.

text Message text, comprised of any string of ASCII characters. The text length is limited to 66 characters *minus* the length of the *session/job identity* under which your (*sender's*) session/job is running. The text is terminated by the end of the record on which it appears, or by a carriage return. The text is optional, but the preceding semicolon delimiter is always required.

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	YES
	Programmatically?	YES
Breakable?	NO	

OPERATION

Transmits a message from the sender's session or job to another session or job currently running. The message appears on the receiving user's standard list device, in this format:

FROM / [*sjname,] username.acctname/text*

sjname }
username }
acctname } The names of the transmitting session/job and user, and the name of the account under which they are running.

text The message text, in ASCII characters.

:TELL

(Continued)

If you send the message to a terminal that is currently interacting with a program, MPE queues the message as high as possible among the current input/output requests but does not interrupt reading or writing in progress. If the session/job or user designated to receive the message is not running, the transmitting session/job receives a system message indicating this. MPE rejects the :TELL command if the complete message (*FROM/ . . . text*) exceeds 72 characters, or if the receiving device is operating in the *quiet mode*. (See SETMSG command specifications.)

EXAMPLE

To send a message to a user identified as BROWN, logged-on under Account A, running a session named BROWNSSES, you could enter:

```
:TELL BROWNSSES,BROWN.A;SYSTEM WILL BE SHUT DOWN AT 4:30 TODAY.
```

TEXT DISCUSSION Pages 3-10, 3-15, 5-5, 9-1 through 9-4.

:TELLOP

Sends message to console operator.

SYNTAX

:TELLOP [<i>text</i>]

PARAMETERS

text

Message text, composed of any string of ASCII characters. Limited to a maximum length of 56 characters. (Optional parameter.)

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	YES
	Programmatically?	YES
Breakable?		NO

OPERATION

Transmits a message from the sender's session/job to the operator's console. The message appears in this format:

MS/time/sjnum/text

time Time message transmitted.

sjnum Session/job number assigned to transmitting session or job.

text Message text.

MPE rejects the :TELLOP command if the message *text* exceeds 56 characters.

EXAMPLE

To send a message to the operator instructing him to mount a particular tape, you could enter:

```
:TELLOP PLS MOUNT MYTAPE, VERSION 1
```

TEXT DISCUSSION Pages 3-10, 3-15, 9-4.

RUNNING SESSIONS

SECTION

III

MPE allows you to conduct interactive sessions through cathode-ray tube (CRT) or hard-copy terminals. During these sessions, you interact conversationally with MPE, entering each command and then receiving an immediate response to it. The following example of a typical session provides an overview of how sessions are run and what they allow you to do. In this session, the programmer is to accomplish these tasks:

1. Create a disc file containing a source program written in FORTRAN language.
2. Compile this source program, prepare it into an object program, and execute the object program.
3. Save the file in which the object program resides, for future use.

If you were running this session, you could proceed as shown below. In this discussion, specific user actions and system responses are keyed to items in the session listing that appears in figure 3-1. User input is underlined.

1. Establish contact with MPE by turning your terminal on (and dialing the computer system if the terminal features a telephone connection), and pressing the *return* key. In response, MPE prompts you for your first command by printing a colon on the terminal display.
2. Enter the `:HELLO` command to request MPE to begin a session (figure 3-1, item 1). This procedure is known as *logging on*. MPE validates your access by verifying that you have entered the correct user and account names and passwords required for your session. MPE also verifies that you have the capability to run sessions, as specified by your Account Manager. Furthermore, it verifies that any limits established for central processor and on-line connect time allowed your account and file group have not already been reached by previous sessions and jobs, and that the system has not already reached the limit set by the operator for the number of sessions that can be run concurrently.
3. If your access is valid, control transfers to the Command Interpreter, which displays a unique session number and other log-on information on the terminal (Item 2). The Command Interpreter also opens two files that are always required by *all* sessions:
 - The *standard input file*, which is equivalent to the input device through which all commands are entered during the session. This device is always the terminal over which you initiate the session.
 - The *standard listing file*, which is equivalent to the output device on which MPE prints all responses to your commands plus all program output not specifically directed to other devices. Except in very rare cases, this device is the same terminal used as the standard input file. (The device used is determined during system configuration, when each potential standard listing device is related to a corresponding potential standard input device.) There is always one standard input file/device and one standard listing file/device for each session.

When the above initialization is complete, the Command Interpreter prompts you (with a colon) for the next command.

```

(1) :HELLO MAC.TECHPUBS
(2) { SESSION NUMBER = #534
      TUE, JAN 27, 1976, 9:02 AM
      HP32002A.00.00
      *** PR-2/30 ***
(3) :EDITOR
(4) { HP32201X.4.3B EDIT/3000 TUE, JAN 27, 1976, 9:02 AM
      /ADD
      1 $CONTROL USLINIT
      2 C FORTRAN EXAMPLE
      3 DO 10 I=1,3
      4 B = I
      5 A = SIN(B) + COS(B)
(5) { 6 WRITE(6,15)A
      7 15 FORMAT ( " VALUE OF A = ",F5.2)
      8 10 CONTINUE
      9 STOP
      10 END
      11
      ...
(6) /KEEP MYPROG
(7) /END
      IF IT IS OK TO CLEAR RESPOND "YES"
      CLEAR? YES
(8) END OF SUBSYSTEM
(9) :FORTGO MYPROG
(10) PAGE 0001 HP32102B.00.0

(11) { 00001000 $CONTROL USLINIT
      00002000 C FORTRAN EXAMPLE
      00003000 DO 10 I=1,3
      00004000 B = I
      00005000 A = SIN(B) + COS(B)
      00006000 WRITE(6,15)A
      00007000 15 FORMAT ( " VALUE OF A = ",F5.2)
      00008000 10 CONTINUE
      00009000 STOP
      00010000 END

(12) { **** GLOBAL STATISTICS ****
      **** NO ERRORS, NO WARNINGS ****
(13) { TOTAL COMPILATION TIME 0:00:01
      TOTAL ELAPSED TIME 0:00:29
(14) END OF COMPILE
(15) END OF PREPARE

(16) { VALUE OF A = 1.38
      VALUE OF A = .49
      VALUE OF A = -.85
(17) END OF PROGRAM
(18) :SAVE $OLDPASS, MYPROGX
(19) :BYE

(20) { CPU (SEC) = 4
      CONNECT (MIN) = 6
(21) { TUE, JAN 27, 1976, 9:08 AM
      END OF SESSION

```

Figure 3-1. A Typical Session

4. Enter the `:EDITOR` command (Item 3) to invoke the Editor subsystem, which you will use to enter your FORTRAN source program. The Command Interpreter passes control to the Editor, which prints a message identifying itself, followed by a slash-mark prompt for your first directive (Item 4).

NOTE

Between the time you invoke a subsystem such as the Editor or a compiler, and the time it prompts you for your first directive, a pause of varying length typically occurs. A similar pause occurs between the time a user program is invoked and the moment it begins execution. This pause reflects the time the Command Interpreter requires to ready and load the subsystem or user program; during this time, the system is performing a complete linkage edit. You can minimize the duration of the pause by asking your System Supervisor to allocate (permanently load) the program.

An MPE subsystem is any program that is run by a unique MPE command, such as `:FORTRAN`, `:SPL`, or `:EDITOR`.

5. Using the Editor `ADD` command, enter the records making up your FORTRAN program, preceded by a `$CONTROL USLINIT` compiler subsystem command to initialize the file onto which the FORTRAN compiler writes its object code (Item 5). The Editor places these records in an empty `TEXT` file which it automatically creates for you. Save a permanent copy of this `TEXT` file under the name `MYPROG` (Item 6); the Editor requests the MPE File Management System to place the file in the MPE Permanent File Directory.
6. Terminate the Editor (Item 7). This operation unloads the Editor, returning control to the Command Interpreter which prints `END OF SUBSYSTEM` (Item 8) on the terminal and then prompts you (with a colon) for a new command.

NOTE

Between the time you finish using a subsystem and the moment you are prompted for the next MPE command, another pause occurs. This reflects the time that the subsystem requires to close all files that it has been using, deallocate all resources such as code and data segments, and terminate itself.

7. Enter the `:FORTGO` command (Item 9) to compile, prepare, and execute your FORTRAN program. The Command Interpreter first invokes the FORTRAN compiler. The compiler prints an identifying message (Item 10), reads your source program from the file `MYPROG` and compiles it, and displays: a listing of your source text with line numbers assigned by the Editor (Item 11); a verification that the program is free of errors and has been successfully compiled (Item 12); and certain compilation statistics (Item 13). Then, the compiler prints `END OF COMPILE` (Item 14) and terminates. Control returns to the Command Interpreter which now invokes the MPE Segmenter.
8. The Segmenter prepares the compiled program into a temporary object program file (which you can later reference under the name `$OLDPASS`). When this task is complete, the Segmenter displays `END OF PREPARE` (Item 15) and returns control to the Command Interpreter which now runs the newly-prepared object program.

9. The object program prints its output on the terminal (Item 16), followed by the notation END OF PROGRAM (Item 17), and returns control to the Command Interpreter. Once more, the Command Interpreter issues a colon prompt for another command.
10. Enter a :SAVE command (Item 18) to change the temporary file \$OLDPASS to a permanent file named MYPROGX, so that you can use your program in future sessions without re-compilation. The Command Interpreter directs the MPE File Management System to place the file in the Permanent File Directory and then issues another colon prompt.
11. Enter the :BYE command to terminate your session (Item 19). This procedure is known as *logging-off*. The Command Interpreter causes the closing of all files that remain open and removes references to temporary files associated with your session; prints the central-processor and connect time accumulated by the session (Item 20); adds these values to separate totals maintained at the account and group levels; prints the current date and time, and the message END OF SESSION (Item 21), and terminates communication with you. Now, the Command Interpreter terminates, returning control to MPE.

LOGGING-ON

To log-on to the system, proceed as follows:

1. Turn your terminal on and set it to the on-line (or remote) mode. *On-line mode* means that all signals generated by pressing keyboard keys are transmitted to the computer rather than simply used for controlling the terminal itself (as in *local mode*).
2. If the terminal is already on-line, check to see if anyone is already using it. If the paper or display screen is blank or shows the words END OF SESSION or END OF JOB on the last line, this indicates that the terminal is free. You can verify this by pressing the *return* key twice: if "ERR 0" appears, the terminal is free; if "ERR 1" appears, the terminal is being used. (If it is in use, select another terminal and return to Step 1.)
3. If your terminal, like the HP 2640A, offers several other control settings, set these as follows:

```
BLOCK MODE .....Off
AUTO L.F. (Line Feed) .....Off
DUPLEX .....FULL
PARITY .....NONE
BAUD RATE .....2400 (equivalent to 240 characters-per-second), or
                    maximum setting below this value. For Type 103
                    modem, set to 300.
```

4. If your terminal is connected directly to the computer (does not use a telephone connection over a switched line), it is ready for use; proceed to step 5, below. But if it has a telephone connection, follow these directions:
 - a. Dial the computer telephone number, which you obtain from your System Manager/Supervisor or Console Operator. The telephone should ring once or twice and then return its carrier signal. For acoustically-coupled equipment, this is a high-pitched tone in the telephone receiver; for other equipment, a CARRIER LIGHT goes on. If you do not receive this signal, the computer may not be in operation.

- b. When you receive the carrier signal, place the telephone receiver firmly in its receptacle (or, for some equipment, press the DATA button). The terminal is now ready for use.
5. Press the return key to generate a carriage return and produce the first prompt character (colon). Then, enter the :HELLO command as described below.

NOTE

Further information on operating a terminal appears under *USING A TERMINAL* at the end of this section.

Although you communicate with MPE through the terminal, you have access to all system peripheral equipment. General procedures for preparing the most common devices appear in table 3-1. Specific requirements, however, may vary between different models of a specific device. See the device reference manuals for details.

INFORMATION ALWAYS REQUIRED

In your :HELLO command, you must always specify:

- A *user name*, which is a valid identity that allows you to log-on. This name is available from your Account Manager, and does not necessarily correspond to a particular real person. In fact, several persons at the same time can log-on using the same user name.
- An *account name*, which identifies the billing entity to which system resources such as central-processor time, on-line connect time, and file space are allocated and charged. This billing entity may be a person, project, department, company, or so forth. A set of files and a list of user names are also associated with the account. Frequently, the users of the account are people who are working together or who are located at a common facility. The account name is established by the System Manager, and may be obtained from him or the Account Manager.

You must supply both a valid user name and account name in your :HELLO command. Otherwise, MPE rejects your log-on attempt and prints an error message to that effect. If your log-on attempt is accepted, however, MPE verifies this by printing specific log-on information and prompting you for your next MPE command. Such information appears in the following example, where a user has logged on under the user name MAC and the account name TECHPUBS:

```

Initial prompt
(received after
entering return)
User name      Account name
:HELLO MAC.TECHPUBS
SESSION NUMBER = #S4
SAT, AUG 30, 1975, 8:39 AM
HP32002A.00.00
MPE software  Update level  Fix level
part number  Version level
: ← Prompt for new command
  
```

Unique session number assigned by MPE

Current date and time session begins execution

Table 3-1. Preparing Input/Output Devices

DEVICE	PREPARATION	SHUT-DOWN
Line Printer	<ol style="list-style-type: none"> 1. Load printer with proper paper or forms needed for your output. 2. Press POWER ON Button. 3. Press START Button. 	<ol style="list-style-type: none"> 1. Press STOP Button. 2. Press PAGE EJECT Button once for each page of output to be removed, and tear this output off following last page. 3. Press POWER OFF Button.
Magnetic Tape Unit	<ol style="list-style-type: none"> 1. Load tape onto tape unit, as directed by Console Operator. 2. Select tape unit number you desire, with appropriate button or dial. 3. Press LOAD Button to move tape to its load point. 4. Press ON LINE Button to connect tape unit on-line. 	<ol style="list-style-type: none"> 1. Press RESET Button to place tape unit off-line. 2. Press REWIND Button to rewind tape to its beginning and unload it. 3. Remove tape from tape unit.
Card Reader	<ol style="list-style-type: none"> 1. Place cards to be read in input hopper. 2. Press POWER ON. 3. Press HARDWARE EOF Button if one appears on this card reader. (This ensures that no other program that owns the card reader can read your cards.) 4. Press RESET Button. 	<ol style="list-style-type: none"> 1. Press STOP Button. 2. Press POWER OFF Button. 3. Remove cards from output hopper.
Card Reader/ Punch/ Verifier	<ol style="list-style-type: none"> 1. Turn Power ON/OFF Switch to ON. 2. Turn ON LINE/OFF LINE Switch to ON LINE. 3. Load cards to be read into Hopper 1. (Hopper 1 is default hopper for reading.) 4. Load cards to be punched into Hopper 2. (Hopper 2 is default hopper for punching.) 5. Press STOP/RESET Button twice. 6. Press START Button once. 	<ol style="list-style-type: none"> 1. Turn ON LINE/OFF Switch to OFF to clear all cards. 2. Turn Power ON/OFF Switch to OFF.

NOTE

The user name and account name combination, sometimes used with an optional session name (described on page 3-10), constitute a *fully-qualified session identity* — the minimum required log-on input.

When you enter the :HELLO command or any other MPE command, always remember to terminate it by pressing *return*. This action transmits the command to MPE for execution.

The *session number* assigned by MPE always uniquely identifies your session to MPE and to other users. MPE assigns such numbers to sessions in sequential order as they are logged-on.

In addition to the standard log-on information, sometimes a welcome message from the Console Operator appears following this display.

By pressing the BREAK key, you can terminate the output of log-on information and immediately obtain the prompt for your next command.

INFORMATION SOMETIMES REQUIRED

In certain instances, you may be required to furnish information in addition to the user and account names in your :HELLO command. This information includes:

- File Group name.
- One or more passwords.
- Terminal type code.

GROUP NAME. Each time you log-on, MPE associates your user name with a particular file group that you use for your local file domain. A group is simply a book-keeping facility that partitions some of an account's file domain and resources into a private sub-domain. The on-line connect-time, central processor time, and disc-file space that you use during this session are charged to this group as well as to the overall account.

The group you select at log-on for your local file domain is known as your *log-on group*. If your account manager has associated a *home group* with your user name, and if you desire this group as a log-on group, you need not specify this — MPE automatically assigns the home group as your log-on group when you log-on. But if you desire to use some other group as log-on group, you must specify that group's name in your :HELLO command, in this way:

```
:HELLO  MAC.TECHPUBS, YGROUP
                        ↑
                        Group name
```

NOTE

If your user name is not related to a home group, you *must* enter a group name in your :HELLO command — otherwise, your log-on attempt is rejected.

Once you log-on, if the normal (default) file security provisions of MPE are in force, you have unlimited access to all files in your log-on and home groups. Furthermore, you can read files and execute programs stored in the public group of your account and the public group of the System account. You cannot, however, access any other files in any way. Further information about files and file security appears in Section VI.

PASSWORDS AND THEIR SECURITY. To enhance the security of an account, and to prevent unauthorized accumulation of charges against the account, the System Manager may assign a password to the account. Similarly, an Account Manager may associate passwords with the user names and groups belonging to his account. If you are using an account, user name, or group (other than your home group) that has a password, you must furnish that password when you log-on. Include the password after the name of the protected entity, separated from that name by a slash mark (/). (In MPE, slash marks denote security.) For instance, if the group XGROUP requires a password and if you wish this group as your log-on group, you could enter the password in this fashion:

```
Group password
      ↓
:HELLO MASON.TECHPUBS,XGROUP/XPASS
SESSION NUMBER = #S2
SAT, AUG 30, 1975, 8:37 AM
HP32002A.00.00
```

NOTE

When specifying your home group as your log-on group, you need not enter a password even if that group has such a password.

If you are working at a facility that uses passwords, you can help ensure that these passwords do not become available to unauthorized users by suppressing printing of the passwords at the terminal as follows:

1. With the terminal in full-duplex mode (manually set by a switch on the terminal), prior to entering the password, disable the MPE Echo Facility by pressing the ESC and ; keys. In full-duplex mode, the terminal does not print characters as they are entered; disabling the Echo Facility prevents MPE from echoing input from the terminal back to the terminal printer.
2. Enter the password, which will not appear on the terminal display.
3. Re-enable the Echo Facility by pressing the ESC and : keys. This again allows your input to be displayed.

More information about transmission (duplex) mode and the echo facility appears under *USING A TERMINAL* at the end of this section.

Sometimes, when logging-on to the system, it is more convenient to have MPE prompt you for any required passwords. You do this by omitting the passwords from the :HELLO command. Then, if you log-on at a hard-copy terminal, the command is printed in the normal way and MPE prompts you for the password, moves the carriage down one line, prints an eight-character mask, and returns the carriage to the beginning of the line, as shown below. When you enter the password, MPE echoes it on top of the mask to ensure privacy. This action is repeated for all passwords required.

```
Mask → :HELLO MASON.TECHPUBS,XGROUP
        GROUP PASSWORD?
        MPEXXXXX
        SESSION NUMBER = #S3
        SAT, AUG 30, 1975, 8:38 AM
        HP32002A.00.00
```



NOTE

This procedure is not effective for a CRT terminal because the password characters *replace* those in the mask as you enter them, and thus become visible to all users. However, if logging-on at a CRT terminal, you can protect your passwords by disabling the Echo Facility as follows:

1. With the terminal in full-duplex mode, enter the :HELLO command.
2. When you press *return*, MPE prompts you for any password required.
3. Disable the Echo Facility as directed above, and enter the password; MPE does not echo the password as you enter it.
4. After your log-on is complete, re-enable the Echo Facility.

In cases when your standard input device and standard listing device are different devices (have different logical unit numbers), MPE *never* echoes the passwords entered. Such cases, however, are rarely encountered in sessions.

TERMINAL TYPES. When you first establish contact with the system, MPE must be able to determine, for its own use, certain characteristics about the terminal you are using to conduct your session. These characteristics include such elements as input speed, output speed, and delay factors for carriage returns, and they depend upon the particular type of terminal used. Thus, MPE can establish the information required if it can determine the terminal type used.

During system configuration, the System Manager or Supervisor determines the type of terminal most frequently used among all hardwired terminals at your site and assigns this as the default terminal type for hardwired terminals. Thus, when you log-on using this type of terminal, MPE already has all the terminal information it needs. But, if you log-on using a different type of hardwired terminal, or a terminal that is not hardwired, you must override the default information by specifying the correct terminal type, selected from the list appearing in the :HELLO command specifications. For instance, if you are logging-on at an HP 2600A but the default terminal type at your site is a 2640A, you would enter:

```
:HELLO MAC.TECHPUBS; TERM=4 ← Terminal type code
```

OPTIONAL INFORMATION

For some sessions, you may want to specify even more information than that already described. This information is purely optional and is normally satisfied by system default values. It includes:

- Session name qualifier.
- Central-processor time limit.
- Execution priority.

SESSION NAME. When several users are logged on under the same user and account names, the only way you, other users, and MPE can uniquely distinguish your session is through the *session number* assigned by the system at log-on. (You will find this type of identification useful when you want to display status information about your session (:SHOWJOB command) or to positively identify yourself as the sender or recipient of messages to or from other users (:TELL) or the Console Operator (:TELLOP and =TELL). As an alternative to the session number, you can also establish an arbitrary session name for this purpose, as follows:

Session name
↓
:HELLO MYNAME,MAC.TECHPUBS; TERM=4

In the above example, the session name MYNAME further qualifies the session identity MAC.TECHPUBS. The session name is usually as satisfactory for its purpose as the session number, and it is normally easier to remember. But, because the session name is purely arbitrary and thus may be coincidentally duplicated by other users, *it is not guaranteed* to be unique in the same way as a session number.

TIME LIMIT. Once you log-on, your session normally runs until you or the Console Operator terminate it, with no limit imposed on the amount of central processor time you can use. Circumstances may arise, however, where a limit on central-processor time may be useful. For instance, suppose you are running a session for the sole purpose of testing a newly-developed experimental program, and do not wish the program to run indefinitely if it encounters an endless loop. You can prevent such looping by placing a limit, at log-on time, upon the amount of central processor time your session can use. You do this in the :HELLO command, specifying the limit in terms of seconds as follows:

:HELLO MAC.TECHPUBS; TIME=600 ← *10-minute central-processor
time limit*

When this limit is reached, the program — and the entire session — are automatically aborted, with an appropriate message.

SCHEDULING. Before your session begins, MPE's Scheduling System establishes the execution priority with which the session competes with other sessions or jobs for access to the central processor. This priority not only applies to the session itself, but is also the default execution priority for all

programs run under the session. The Scheduling System places the session in its *Ready List*, which is ranked in order of execution priority. When a session, job, or program in execution is completed, terminated, or interrupted, the MPE Dispatcher searches the Ready List for the session, job, or program of highest priority awaiting execution. When this entity is your session, MPE transfers control to it and processing begins on your behalf.

Normally, the standard priority assigned by MPE as a default will be adequate for your session. But when you do require a greater or lesser priority, you can override this default by specifying the priority in the :HELLO command. You can select any of the four classes described in table 3-2. One of these classes (BS) dictates that your session is dispatched in a *linear* manner. This means that the session accesses the central processor when it achieves the highest priority in the system, and continues to run until it is completed, prematurely terminated, or suspended to await input/output. The remaining priority classes (CS, DS and ES) specify that the session is dispatched in a *circular* manner. This means that the session competes with other sessions or jobs for the central processor, and once it attains this access, runs until completed, prematurely terminated, suspended while awaiting input/output, or until a time-quantum (interval of limited duration established during system configuration) expires. At this time, control passes to the session or job of next highest priority. The default priority class assigned for sessions is CS.

Table 3-2. Session/Job Priority Classes

CLASS		DESCRIPTION
High ↑ ↓ Low	BS	Highest priority requestable at session or job level. Linear class.
	CS	Standard priority, used as default for interactive sessions and for all programs run under them. Circular class.
	DS	Standard priority, used as a default for multiprogrammed batch jobs and for all programs run under them. Circular class.
	ES	Lowest priority requestable at session or job level, allowing you to run sessions and jobs of very low priority. If no sessions or jobs are using this or any higher priority class, the central processor becomes idle. Circular class.

Suppose, as an example, that you have a session that you want to run at lower priority than all the others automatically assigned priority-class CS. You can request the DS class in your :HELLO command, as follows:

```
:HELLO MAC•TECHPUBS; PRI=DS
```

Your session then runs in the background, acquiring the central processor only when no session, job, or program of CS priority or higher is active in the system.

NOTE

When the System Manager creates an account, he specifies a maximum limit on the priority which you can request when running under the account. Similarly, when the Account Manager creates the list of user names under which the account can be accessed, he associates with each name a maximum priority that anyone using that name can request. When you log-on, your account's maximum priority is checked against your user name maximum priority, and you receive the lower of these as your maximum limit.

When you log-on, MPE assigns your session a numeric *input priority* that is used to screen your session against a *job fence* that may be established by the Console Operator to limit system access to certain sessions and jobs. The input priority assigned is 8 if logging (recording) of session/job initiation by MPE is enabled, or 13 if this logging is disabled. The job fence applies to both sessions and jobs, and is also a numeric value; it restricts access to sessions/jobs whose input priorities exceed the job fence value. Thus, if your session has an input priority less than or equal to the current job fence set by your Console Operator, your session is denied access and the following message appears on the terminal:

```
CAN'T INITIATE NEW SESSIONS NOW
```

To successfully log-on, you must re-enter the :HELLO command with a higher input priority or wait until the Operator lowers the job fence. In the :HELLO command, you can request priorities ranging from 1 (lowest) to 13 (highest) in this way:

```
:HELLO MAC.TECHPUBS; INPRI=12
```

Request for input priority 12 ←

NOTE

Because the default input priority for a session is typically 8, you usually need to specify an input priority in your :HELLO command only when the current job fence is greater than or equal to 8, but less than 13.

The message CAN'T INITIATE NEW SESSIONS NOW also appears if you attempt to exceed the maximum number of sessions concurrently allowed on the system, as defined by the Console Operator.

If you have the System Manager or Supervisor user capability, you can request the highest possible input priority for the session, as follows:

```
:HELLO MAC.TECHPUBS; HIPRI
```

The HIPRI entry causes your session to be scheduled regardless of the current job fence or limit on the number of sessions that can be executed (as defined by the Console Operator command = LIMIT).

ERRORS DURING LOG-ON

If you are logging-on through a terminal, you must complete that procedure within a time-limit specified during system configuration. Typically this limit is two minutes. If you fail to log-on within this limit, MPE disconnects your terminal and you must once again establish contact with the system.

If the connect-time, central-processor time, or file-space limit allotted to your account or group has been exceeded by previous sessions and jobs, your session is rejected at log-on. You must then ask the System Manager to adjust or clear these limits and try to log-on again. If these limits expire during a session, however, the session is allowed to continue until you terminate it; the next session or batch job attempted under this account or group, however, is rejected at log-on.

NOTE

Do not confuse this central processor time-limit with the time-limit specified in the *TIME-cpusecs* parameter of the *:HELLO* command. The central-processor time-limit for accounts and groups limits the total central-processor time that can be accumulated collectively by all sessions and jobs that have run under the particular account or group — it is checked only at log-on time, and can only terminate access at that time; it will never cause a session to abort. However, the *TIME=cpusecs* parameter (or its default value) applies to each individual session, and aborts the session as soon as this limit is reached.

If you enter an illegitimate *:HELLO* command (with an incorrect user, account, or group name or a bad password, for instance), your session is rejected. Errors also result when you attempt to log-on without having the user capability required to run sessions; when you use an incorrect delimiter in your command parameter list; when you specify an improper central processor time limit; or when you request an improper priority.

Most command errors that occur during log-on are accompanied by error messages of this format:

ERR *errornumber*[,*parameternumber*]

The *errornumber* indicates the MPE number relating to the error description shown in Section X of this manual. The *parameternumber* appears only when the error can be directly attributed to a particular command parameter; it denotes the sequential number of the parameter in the parameter list.

As an example, in response to the most common type of error — an illegitimate *:HELLO* command — the following message appears:

ERR 8

With this type of error, the erroneous element in the *:HELLO* command is not identified.

NOTE

Although you can obtain an explanation of an error number output during a session by entering any character other than *return* following the error number, you cannot do this with error numbers output *before* your log-on is accepted. Instead, such error numbers are followed immediately by a prompt for a new :HELLO command. For an explanation of the error, turn to Section X of this manual.

All :HELLO commands transmit a message to the Operator's Console. Legitimate :HELLO commands duplicate on the Console the normal log-on message received by the user, showing session name, number, and log-on time. Illegitimate :HELLO commands are also echoed to the Console for logging purposes.

LOGGING-OFF

To log-off from the system, enter the :BYE command. In response, MPE prints a standard log-off display and terminates communication with you. The log-off display shows the central-processor time (in seconds) and terminal connect-time (rounded upward to the nearest minute) used by your session, the current date and time, and the message END OF SESSION, as follows:

```
:BYE

CPU (SEC) = 1
CONNECT (MIN) = 1
SAT, AUG 30, 1975, 8:40 AM
END OF SESSION
```

MPE also adds the central-processor time and connect-time, along with the file space used by your session, to the resource-usage counters maintained for your log-on account and group. During your session, you can determine the account and group totals by entering the :REPORT command (Section V).

If you hang up the receiver prior to logging-off at a terminal with a telephone connection, MPE automatically terminates your session by implicitly issuing a :BYE command.

If you enter a :HELLO command before logging-off, MPE terminates your current session and immediately initiates a new one. If you are logged-on at a terminal with a telephone connection, MPE does not disconnect the terminal. Instead, MPE maintains the connection, allowing the new session to begin immediately.

INTERRUPTING COMMAND EXECUTION

While executing an MPE command, you may suddenly need to perform some other function. For instance, you may need to send a message to another user (:TELL command) or the Console Operator (:TELLOP), determine information about other sessions and jobs (:SHOWJOB), create a new disc file (:BUILD), delete a file (:PURGE), or list the files in your account or group (:LISTF). To perform any of these functions, you usually must first abort or suspend the current command operation by striking the BREAK key on your terminal keyboard. (On some terminals, this key may be labeled *INTERUPT* or *ATTENTION*.) The exact effect of pressing the BREAK key depends upon whether the operation you interrupt is a command that does not execute a system or user program (*non-program command*) or one that does execute such a program (*program command*).

NOTE

Not all MPE command operations respond to the BREAK request. You can determine which operations can be interrupted by consulting the *BREAKABLE?* entry in the command specifications, as well tables 3-3 and 3-4 in this section.

Table 3-3. Non-Program Commands

BREAKABLE	NON-BREAKABLE	
:BYE	:ABORT	:PURGE
:HELLO	:ALTSEC	:RELEASE
:LISTF	:BUILD	:RENAME
:REPORT	:COMMENT	:RESET
:RESTORE	:CONTINUE	:RESETDUMP
:SHOWDEV	:DATA	:RESUME
:SHOWIN	:EOD	:SAVE
:SHOWJOB	:EOF:	:SECURE
:SHOWOUT	:EOJ	:SETDUMP
:STORE	:FILE	:SETMSG
:STREAM	:FREERIN	:SHOWTIME
	:GETRIN	:SPEED
	:JOB	:TELL
	:PTAPE	:TELLOP

Table 3-4. Program Commands (All Breakable)

:BASIC	:PREP
:BASICCOMP	:PREPRUN
:BASICGO	:RJE
:BASICPREP	:RPG
:COBOL	:RPGGO
:COBOLGO	:RPGPREP
:COBOLPREP	:RUN
:EDITOR	:SEGMENTER
:FORTGO	:SPL
:FORTPREP	:SPLGO
:FORTRAN	:SPLPREP

INTERRUPTING NON-PROGRAM COMMANDS

Most MPE commands do not result in the execution of either system or user programs; some of these, such as :LISTF (to list file characteristics) and :STORE (to store disc files on magnetic tape), are breakable. (A list of all breakable and non-breakable non-program commands appears in table 3-3.) When you wish to enter another MPE command while a breakable non-program command is being executed, strike the BREAK key. This terminates (aborts) the current command. MPE then prints a colon to prompt you for the next command.

NOTE

MPE may not respond instantaneously to your BREAK request, perhaps because of the current system load or delays inherent in the BREAK operation. When this occurs, it is *not* necessary to strike the BREAK key more than once to initiate the BREAK sequence — MPE will respond to your initial BREAK request as soon as possible.

INTERRUPTING PROGRAM COMMANDS

Program commands invoke MPE subsystems or run user programs. For instance, the :SPL command invokes the SPL compiler subsystem and is equivalent to issuing a command to run that subsystem; other examples of commands that execute subsystems are :COBOL (to run the COBOL compiler), :FORTRAN (to run the FORTRAN compiler), and :EDITOR (to run the Editor). The :RUN command is used to run user programs. All program commands are breakable; a complete list of them appears in table 3-4.

When you press the BREAK key to interrupt one of these program commands, the execution of that command is suspended and the Command Interpreter issues a prompt for a new MPE command. If you then enter a *non-program command* (other than :HELLO, :BYE, :JOB, or :DATA), the Command Interpreter performs the requested operation and then allows you to re-activate the suspended program by entering :RESUME. But, if you enter another *program command* (such as :FORTRAN, :EDITOR, or :RUN) or one of the non-program commands :HELLO, :BYE, :JOB, or :DATA, the Command Interpreter prints the following message on your terminal:

ABORT?

If you respond *YES* to the ABORT? message, the Command Interpreter aborts the current program and executes the command you entered, in the usual way.

If you respond *NO* to the ABORT? message, the Command Interpreter prints the message NOT ALLOWED IN BREAK and prompts you for another command. If you now enter :RESUME, the suspended program resumes at the point where it was interrupted.

To illustrate how the BREAK function may be used, consider the following example.

1. When you run the program MYFILEX, the first record output contains a value that you know is in error. You wish to inform another user that this program is not producing valid data (by entering the :TELL command), and then to resume your program to check subsequent output.

```
:RUN MYFILEX                                Request to run MYFILEX.
```

```
FIRST ANSWER:      1.000 ←—— Erroneous value.
```

2. Suspend the program by pressing the BREAK key, returning control to the Command Interpreter.
3. When the Command Interpreter prompts you for a command, enter the :TELL command in this manner to send your message:

```
:TELL BOB.TECHPUBS; MYFILEX HAS LOGICAL ERROR.
```

4. The :TELL command is not a program command and therefore does not require you to abort the suspended program. Thus, MPE transmits your message and prompts you for another command.
5. Request resumption of the suspended program by entering the :RESUME command. The program will resume execution at the point where it was interrupted.

```
:RESUME
```

```
SECOND ANSWER:  1000000.000 ←—— Next record output by MYFILEX program.
```

As another example:

1. You decide to suspend the current program (MYFILEX) by pressing the BREAK key. When you receive a prompt for a new MPE command, you request execution of another program (MYP-ROGX):

```
:RUN MYFILEX
```

```
FIRST ANSWER:      1.000 ←—— Output from MYFILEX,  
:RUN MYPROGX      followed by BREAK.
```

2. Because the second :RUN command executes a new program, MPE prompts you to abort the suspended program, effectively terminating the break. Respond YES to abort the current program and execute the new program (as shown below) or NO to return control to the Command Interpreter:

```
ABORT? YES
```

```
ERR 2 ABNORMAL PROGRAM TERMINATION ←—— MYFILEX terminates.
```

```
VALUE OF A =  1.38  
VALUE OF A =  .49  
VALUE OF A = -0.85  
END OF PROGRAM
```

} MYPROGX runs,
producing this output.

When you terminate a break by entering :RESUME, the suspended operation normally resumes at once. But if the terminal was waiting for input when you pressed the BREAK key, you will be prompted for this input by the following message when you enter :RESUME:

READ PENDING

You must enter the required input before the suspended operation can resume. Any characters that you entered before pressing BREAK are considered part of the total input string.

NOTE

User programs are initiated with the BREAK facility enabled. But since you may not wish to let those who run your programs interrupt them under certain circumstances, you may programmatically disable the BREAK facility by using the FCONTROL intrinsic. For example, application programs can restrict user access to the Command Interpreter by disabling the BREAK function.

ABORTING A PROGRAM

Sometimes, output from a program you are running may indicate that the program contains logical errors (such as an infinite loop or improper data-manipulating operations). These errors, though not fatal to the program, invalidate the data generated. Thus, you wish to promptly abort the program, correct it, and re-run it later. The following example shows how to do this:

1. When the program MYFILEX, used in the previous example to illustrate the BREAK function, produces the erroneous value shown below, you decide to abort the program.

:RUN MYFILEX

Request to run MYFILEX.

FIRST ANSWER: 1.000 ← *Erroneous value.*

2. Suspend the program by pressing the BREAK key.
3. When MPE prompts you for a new command, enter:

:ABORT

This command terminates the program but in no way disrupts the session. MPE confirms this termination by displaying the ERR 2 error message. (If you request further explanation of this message, you receive the output shown on the last line below.)

ERR 2 X ← *Request for error explanation*
ABNORMAL PROGRAM TERMINATION ← *Error explanation*

4. MPE then prompts you for a new command.

You can also use the BREAK/:ABORT sequence to abort certain MPE subsystem and command operations. Those operations are indicated in the associated command specifications, under the BREAKABLE? entry. (Some of them abort immediately upon entering BREAK, without requiring the :ABORT command.)

READING DATA FROM STANDARD INPUT DEVICE

In a session, you may use the standard input device (terminal) to enter data as well as MPE commands. When you run a program that reads data directly from the terminal and depends upon detecting the end of data to halt the read operation, you must indicate the end of data by entering an :EOD command after the last data record. As an example, when you run the program PARTS, that program prompts you to enter a series of records containing part numbers for an inventory:

```
:RUN PARTS
```

```
ENTER PART NUMBERS
```

You may enter as many part numbers as you need, delimiting this input with :EOD. The program then continues execution, printing the message shown on the last line below at this time:

```
10023  
10027  
10072  
10073  
11005  
11007  
:EOD
```

```
PARTS INPUT COMPLETE
```

```
•  
•  
•
```

NOTE

Although in most cases programmers use :EOD for delimiting data, any record beginning with a colon will delimit the data. The ramifications of using a command other than :EOD for this purpose, however, depend upon whether the standard input file is opened with the file name \$STDIN or \$STDINX, as discussed in Section VI; these ramifications, can have a very important impact upon your session.

You also use the :EOD command to delimit data entered from other data-accepting devices, such as paper tape and card readers. This data is submitted via the :DATA command, as described on page 6-81.

In a session, you do not normally input source programs to a compiler directly from the terminal; instead, you usually direct the compiler to read these programs from text files on disc, which you establish by using the MPE Editor. But, when you do not specify a source file for a compiler, the compiler attempts to read the source file directly from the standard input device (terminal). Frequently, the source program contains a statement (such as END. in SPL) that tells the compiler when to stop reading source records and terminate the compilation. But when using a compiler language that does not provide this convention, you must enter :EOD after the last record of any source program entered from the terminal to ensure proper delimiting of your source record input.

NOTE

The :EOD command is not required when using the BASIC Interpreter, since this subsystem provides different conventions for delimiting data.

FOR PROGRAMMERS: The :EOD command causes the READ or FREAD intrinsic to return the CCG condition code to the calling program. This code indicates the end-of-file condition on the terminal.

EFFECTS OF SPOOLING

Except for discs, MPE manages each input/output device so that only one session or job at a time can have access to that device. Thus, if you are running a program that writes output to a line printer that is being used by another session, your session must wait until the line printer is free. When *spooling* is in effect, however, sessions can write output to temporary files on disc that will later be transmitted to the appropriate peripheral devices. Thus, if the line printer is a *spooled output device*, MPE automatically copies the output to a disc file and allows your session to continue immediately; when the printer is free, MPE then transmits the output from the disc file to the printer.

In a session, the main advantage of spooling is that, by buffering output to disc for later transmission to the destination device, contention among several sessions/jobs for that destination device is reduced. (*Spool* is an acronym for *Simultaneous Peripheral Operations On-Line*.) Your session need not wait for the required device to become available — it can perform its output immediately. Output spooling is permitted for line printers, card punches, and graphic plotters.

Input spooling is also possible in a session. When this is done, input is spooled from the originating peripheral device onto disc, where it is read by your session. This frees the originating device for use by other sessions and jobs. Input spooling is permitted for card readers and magnetic tape units. Further information about input spooling appears in Section IV.

Spooling makes the most efficient use of relatively slow unit-record devices, so that input from or output to those devices becomes independent of your session (and the time required is not charged to the session). Spooling is controlled by the Console Operator, who can determine at any time which allowable devices are spooled. Then, when you run a program or MPE subsystem that writes output to

a spooled device, MPE automatically directs that output to a disc file. The disc file is called a *spooled devicefile* (a file whose ultimate destination in this case is a card punch, line printer, or plotter) that enters the OPENED state the moment that output to it begins. When the program or subsystem completes its output and closes the devicefile (and this file is opened by no other programs in the session), the devicefile enters the READY state. This means that the file is ready for transfer to the destination device. When the destination device is free, MPE selects the spooled devicefile with the highest priority currently pending and copies the file to that device. (Any special printer forms required are requested from the Operator at this time.) During this copying, the devicefile is in the ACTIVE state. Spooled output devicefiles are independent of the session listing and may be completely printed or punched either before the session ends or at some point thereafter.

NOTE

A spooled output devicefile is not transferred from disc until the subsystem or user program that generated the output closes the devicefile. Thus, if a listing produced by the Editor was written to a spooled line printer, it would only appear on that device after the Editor closed that output devicefile.

Spooling has very little apparent impact upon your session. From your point of view, you cannot generally distinguish access of a spooled devicefile from access of the actual destination device. For instance, at a given instant, a file name may refer to a temporary disc file or an actual line printer. Also, when you request information about the file's characteristics, such as record size or device type, you receive the same information whether the file is on disc or copied to a destination device. You can determine which files used by your session are actually spooled devicefiles and what states they are in, by entering the :SHOWOUT command for output files or the :SHOWIN command for input files. These commands and further information about devicefiles are discussed on pages 8-8 and 8-13.

NOTE

In a spooling environment, if an output devicefile runs out of available disc space, no new spooling will occur and the message (SPACED OUT) appears on your output listing (if this listing is directed to a line printer). See your Console Operator for corrective action.

The Console Operator can, at his option, delete a READY spooled devicefile, destroying the output. In the event of a shut-down initiated by the Console Operator or caused by a system failure, spooled output files are saved and can still be used if the Console Operator re-starts the system by using the WARMSTART procedure. This procedure places all spooled output files in the READY state (regardless of their state prior to shut-down). Files that were OPENED will not be complete, but all spooled output files will be printed on their destination devices according to the output priorities established by the requestor. Those that were partially-printed upon failure will be completely printed upon WARMSTART.

PREMATURE SESSION TERMINATION OR SUSPENSION

Under certain circumstances, MPE terminates or suspends your session before you issue the :BYE command. For instance, MPE immediately terminates the session if:

1. The Console Operator issues the = ABORTJOB command (to terminate the session), the = LOGOFF command (to abort all executing sessions and jobs, and prevent further logging-on), or the = SHUTDOWN command (to shut down the system).
2. You enter a second :HELLO command (to begin another session), a :JOB command (to begin a batch job from the terminal), or a :DATA command (to establish the terminal as a data file that can be acquired by other sessions and jobs). All these commands result in an implicit :BYE command, with accompanying log-off output.
3. MPE encounters a hardware end-of-file (or the :EOF: command, used to simulate this) on the standard input device.
4. Your session exceeds the central-processor time limit established by the *cpusecs* parameter of the :HELLO command.

When a system termination (whether normal or abnormal) occurs, all sessions executing at the time are also terminated and must later be re-initiated from the beginning. (All batch jobs are also terminated unless they are spooled jobs designated as *re-startable*, in which case they are temporarily suspended until the Console Operator re-initiates the system with the WARMSTART option as discussed in Section IV.)

NOTE

Because a *power failure* generally has a temporary rather than a permanent impact on sessions and jobs, as described below, it is regarded as a system *suspension* rather than a system *termination*.

When a system power failure occurs, MPE preserves the operating environment prior to complete loss of power. When power returns, the message POWER FAIL appears on your terminal and normal system operation resumes, with interactive sessions on hard-wired terminals and batch jobs continuing at points where they were interrupted — with the possible exception that the last few characters of input/output on the terminal might not have been transmitted. (If any characters remain to be transmitted, enter the X^c control character to delete the line and then re-enter the characters. This control character and others are discussed fully under *Special Keyboard Functions* later in this section.) Sessions initiated on dial-up terminals, however, must be re-initiated because MPE provides no method of holding the connection during the interruption.

When your session directs output to files on a spooled output device, certain commands issued by the Console Operator may prevent or suspend output of those files even though the session itself is not terminated or interrupted. These commands are: = DELETE (to delete READY spooled devicefiles); = SPOOL *1dn*,STOP,DELETE (to terminate output spooling immediately and delete the ACTIVE spooled devicefile); = SPOOL *1dn*,WAIT (to suspend spooling after completing the ACTIVE devicefile); and = SPOOL *1dn*,DEFER (to change the output priority of the ACTIVE spooled devicefile, and return it to the READY state).

ERROR-HANDLING

If you make an error while entering a command in an interactive session, MPE suppresses execution of that command, displays an error number, and returns control to your terminal. The error number is an index number that points to an error message and description appearing in Section X of this manual. When the error can be directly attributed to a particular command parameter, the error number is followed by a number that denotes the sequential location of the bad parameter in the parameter list. For example, the following `:RUN` command produces the error number 108, which points to the message `NON-EXISTENT FILE` in Section X. The number 1 implies that the error resulted from the first (and in this case the only) parameter in the command, which names a file that does not exist.

```
:RUN MYPROGY  
ERR 108,1  
      ↑      ↑  
Error number Parameter number
```

If you require output of the full error message, enter any character other than *return* immediately after the error number. (In the following example, *X* is used.) In response, MPE displays the appropriate error message on the next line and prints a colon to prompt you for another command:

```
:RUN MYPROGY  
ERR 108,1 X  
NON-EXISTENT FILE ← Message  
:  
↑  
Prompt
```

If you do not require the complete error message, enter *return* directly after the error number. MPE then prints a colon, prompting you for another command:

```
:RUN MYPROGY  
ERR 108,1  
:  
↑  
Prompt
```

When certain unusual but non-fatal conditions arise during interpretation or execution of a command, MPE executes the command (if possible), issues a warning message following the command, and allows the session to continue by prompting you for another command, as follows:

```
:TELL HAL.TECHPUBS; DO NOT RUN MYPROG UNTIL TOMORROW.  
USER NOT ACCEPTING MESSAGES  
:  
↑  
Prompt
```

↑
Warning message

A full discussion of Command Interpreter error and warning messages appears in Section X.

USING A TERMINAL

MPE supports various kinds of hard-copy and CRT terminals, all noted in Appendix A. General rules and guide-lines common to most of these terminals are discussed below. More specific rules and descriptions of features unique to a particular terminal appear in the reference manual covering that device.

All terminals are connected to the system over terminal controllers, each of which can support up to 16 terminals. The drivers for the terminals are furnished as part of MPE.

LOG-ON CHARACTERISTICS

MPE supports terminals that run at input/output speeds ranging from 10 to 240 characters per second (cps). When you log-on, MPE senses the speed at which your terminal is set, and interfaces with it accordingly. At any point after log-on, you can alter the input or output speed by entering the :SPEED command described on page 9-5, or by programmatically calling the FCONTROL intrinsic discussed in *MPE Intrinsic Reference Manual*.

When you log-on, MPE assumes that certain operational characteristics are in effect, as described below. If you are running a program that has opened the terminal/file, you can programmatically enable or disable these characteristics by invoking the FCONTROL intrinsic. For further information, see *MPE Intrinsic Reference Manual*.

- *Echo Facility*. This option directs the system to transmit (echo) input from the terminal keyboard back to the terminal printer or display screen. Various ramifications of this option are discussed under *Echo Facility and Transmission Mode* later in this section. *Log-on Setting: Enabled*.
- *System Break Function*. This option permits you to suspend a user program or MPE subsystem, returning control to the Command Interpreter, by pressing the BREAK key on the keyboard. (An MPE subsystem is any program that is run by a unique MPE command, such as :FORTRAN, :SPL, or :EDITOR.) *Log-on Setting: Enabled*.
- *Subsystem Break Function*. This function lets you terminate a specific subsystem function or operation, returning control to the subsystem, by entering Y^c. *Log-on Setting: Disabled*.
- *Parity-Checking*. With this function, MPE checks the parity of data received against the parity computed by the asynchronous channel multiplexor; if a parity error occurs, an error code is made available which you obtain through the FCHECK intrinsic. *Log-on Setting: Disabled*.
- *Tape Mode*. This mode tells the system that input is coming from the paper tape reader. The special characters Line Feed (for spacing one line), H^c (for deleting a character), and X^c (for deleting a line) are handled differently to conform with the tape medium; Line Feed does not actually generate a line feed, X^c deletes the entire line input with the system expecting a carriage return before it accepts more data; and H^c deletes the last character entered but disables output of the backslash that verifies deletion. These special characters are discussed under *Special Keyboard Functions* later in this section. *Log-on Setting: Disabled*.
- *Non-Standard Line Terminator*. This option establishes a character other than *return* as the line (record) terminator. The specified character terminates the record but does not cause the carriage to return. *Log-On Setting: Disabled (return is recognized as the line terminator)*.

ECHO FACILITY AND TRANSMISSION MODE

As you enter information from your terminal keyboard, two factors — echo facility and transmission mode — determine whether or not that information also appears on the terminal's display (print-out or CRT screen). The *echo facility*, a function of the system hardware controlled by MPE, determines whether or not each input character you enter from the keyboard is transmitted (echoed) back to the terminal's display by MPE. The *transmission mode*, a function of the terminal, determines whether or not the *terminal itself* prints input from the keyboard upon its own display as you enter that information. (In half-duplex transmission mode, the terminal prints the input; in full-duplex transmission mode, it does not.) What you actually see upon the terminal display depends upon the interaction of both these factors, as explained below and shown in figure 3-2.

When the *echo facility is on*, the terminal input is echoed to the display by MPE. If the terminal is operating in *full-duplex mode*, the echoed information appears as normal printed lines. If the terminal is in *half-duplex mode* and is not an HP 2640 or 2644, however, the echoed printing is illegible — as you enter the input on such terminals, it is simultaneously printed by the terminal and then subsequently overwritten by the echoed information from MPE, resulting in garbage. (On an HP 2640 or 2644 terminal in *half-duplex mode*, however, the characters are duplicated during echoing so that any character entered appears twice.)

When the *echo facility is off*, input read from the terminal is not echoed by MPE. If the terminal is operating in *full-duplex mode*, no printing appears. If the terminal is in *half-duplex mode*, the input is copied by the terminal itself, and appears as normal, printed lines. (Bear in mind that the only way printing can be suppressed is with the echo facility *off* and the terminal in *full-duplex mode*, as illustrated in figure 3-2.)

When you log-on, the echo facility is always *on* for your terminal. You can turn it *off* by pressing the *ESC* key followed by the *semi-colon* key:

ESC;

You can turn the echo facility back *on* by pressing the *ESC* key followed by the *colon* key:

ESC:

You can programmatically switch the echo facility on and off by using the FCONTROL intrinsic discussed in *MPE Intrinsic Reference Manual*.

Where a terminal can operate in either full- or half-duplex transmission mode, you select the mode by using the appropriate switch on the terminal.

SPECIAL KEYBOARD FUNCTIONS

Once you turn the terminal on, you operate it much as you would a typewriter. In addition to the functions available on a typewriter keyboard, however, there are certain others that handle special operations frequently required in communicating with a computer. Those functions common to most terminals, and the keys most typically used to request them, are listed in table 3-5.

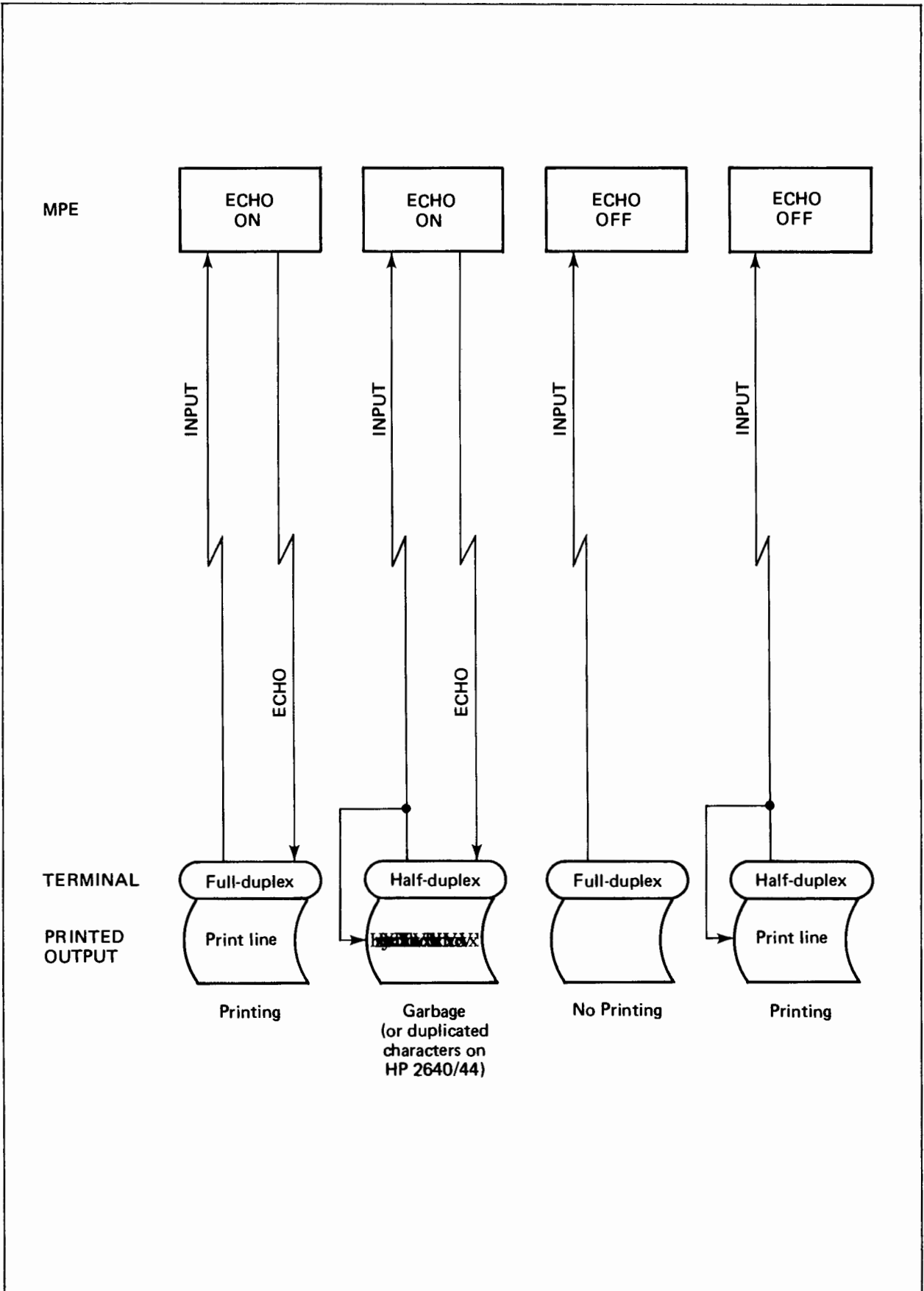


Figure 3-2. Echo Facility/Transmission Mode

Table 3-5. Special Functions and Keys

FUNCTION	KEY
<p>DELETE PREVIOUS CHARACTER</p> <p>To delete <i>n</i> characters, enter H[°] <i>n</i> times.</p> <p>NOTE</p> <p>The superscript ° appearing in this table denotes a control character entered while pressing the CTRL or CNTL key. Thus, to delete three characters, hold down the CTRL (or CNTL) key, strike H three times, and release the CTRL (or CNTL) key.</p>	H [°]
<p>DELETE ENTIRE LINE BEING TYPED</p> <p>The system responds by printing a triple exclamation point, returns the carriage or cursor to the beginning of the line, and then spaces down one line. You may then re-enter the line. The prompt is not repeated.</p>	X [°]
<p>REQUEST SYSTEM BREAK, SUSPENDING CURRENT PROGRAM AND RETURNING CONTROL TO MPE</p>	BREAK
<p>REQUEST SUBSYSTEM BREAK, TERMINATING CURRENT OPERATION AND RETURNING CONTROL TO SUBSYSTEM RUNNING</p> <p>To request this function, the terminal must <i>not</i> be in tape mode.</p>	Y [°]
<p>TERMINATE CURRENT RECORD AND RETURN CARRIAGE OR CURSOR TO FIRST POSITION OF NEXT LINE</p>	RETURN
<p>SPACE TO NEXT LINE AND RETURN CARRIAGE OR CURSOR TO FIRST POSITION OF THAT LINE</p> <p>Because the line-feed character is not transmitted to the terminal's input buffer, multiple lines can be entered in response to a single read request; the length of that response is not limited by the width of the terminal carriage or display screen, but by the way the reading program makes the request.</p>	LINE FEED
<p>PLACE TERMINAL IN TAPE MODE, SO THAT PAPER TAPE INPUT CONTAINING H[°] AND X[°] CHARACTERS CAN BE READ</p> <p>Tape mode is only required to read tapes containing these characters.</p>	Q [°]
<p>REMOVE TERMINAL FROM TAPE MODE</p> <p>To request this function, terminal must be in tape mode; otherwise, your request is treated as a subsystem break request (if this break function is enabled).</p>	Y [°]
<p>PLACE TERMINAL IN ECHO-ON MODE SO THAT CHARACTERS INPUT ARE ECHOED TO TERMINAL PRINTER BY MPE</p>	ESC;
<p>PLACE TERMINAL IN ECHO-OFF MODE SO THAT CHARACTERS INPUT ARE NOT ECHOED BY MPE</p>	ESC;

MPE recognizes the control characters X^c, H^c, Q^c, and Y^c even when they follow an ESC key entry. However, entry of ESC followed by any character (other than one of these control characters or a colon or semi-colon) is interpreted as a two-character string.

NOTE

The character-correction mechanism (H^c) works in one of the following ways, depending upon the type of terminal you are using:

- **CRT Terminals**

All currently-supported CRT terminals can physically back-space the cursor. Therefore, H^c back-spaces the cursor one position, leaving it placed at the position of the character to be replaced. This back-spacing does not erase the character from the screen, but it does delete the character from MPE's internal buffer. As an example, if you enter the following command and then strike H^c seven times, the cursor is positioned as shown:

```
:RUN  PROGR
      ↑
      Cursor position
```

When you enter new characters, the original characters are deleted from the screen.

- **Hard-Copy Terminals**

For those terminals which have physical back-space capability, H^c back-spaces one position and (if the previous character was not also H^c) spaces down one line. This permits you to resume typing beneath the first character to be replaced.

For terminals which have no physical back-space capability, no back-spacing takes place. Instead, H^c echoes a back-slash for each character deleted (unless the terminal is in tape mode). For instance, if you enter H^c seven times after typing the erroneous :RUN command shown below, the following output appears:

```
      Logical cursor position
      ↑
:RUN  PROGX\\ \\ \\ \\ \\ \\ \\
                                     ↑
                                     Actual cursor position
```

You resume typing after the last back-slash.

The system determines what kind of terminal you are using by the *TERM=termtype* parameter of the :HELLO command. If, however, your terminal is hardwired to the system and you specify no terminal type at log-on, your terminal is assigned a default type that is determined during system configuration. Thus, a normally-connected (hardwired) HP 2640A or 2644A terminal typically receives a default terminal type of *11* even if you do not specify this type in the :HELLO command.

SPECIAL TERMINAL TYPES FOR HP 2640 AND HP 2644 TERMINALS

When you log-on at an HP 2640 or 2644 CRT Terminal, to ensure most advantageous terminal operation, you must specify either ;*TERM=10* or ;*TERM=11* in your :HELLO command. (You need not do this, of course, if the default terminal at your site is type 10 or 11.) The option you select depends upon the mode in which you plan to operate your terminal most of the time during the session:

- *Character Mode*, where the information you type is transmitted to the computer character-by-character. (In other words, each character is transmitted as it is typed.) All hand-shaking operations required between the terminal and the computer, and all concerns such as terminal buffer overflow, are managed automatically by MPE.
- *Block Mode*, where the information you enter is transmitted to the computer in lines or blocks of characters, allowing local editing prior to transmission.

CHARACTER MODE (;*TERM=10*). If you plan to run your terminal primarily in character mode, enter ;*TERM=10* in the :HELLO command when you log-on. Although not primarily designed for block mode, ;*TERM=10* does permit you to dynamically switch to that mode during your session by:

1. Turning off the MPE echo facility (by using the ESC and ; keys or the FCONTROL intrinsic).
2. Pressing the BLOCK MODE key, locking it down.
3. Placing the cursor at the first position of the current line and typing the characters in that line, also performing any editing desired.
4. Placing the cursor at the first character to be transmitted.
5. Pressing the ENTER key to transmit the line.
6. Releasing the BLOCK MODE key to return to character mode.
7. Turning on the MPE Echo Facility (by using the ESC and : keys or the FCONTROL intrinsic).

NOTE

If you try to use block mode without first turning off the echo facility, the cursor returns to the beginning of the current line the moment you press the ENTER key. Thus, you lose control of the cursor.

BLOCK MODE (;*TERM=11*). If you plan to frequently switch between block and character mode, enter ;*TERM=11* in the :HELLO command. Then you can use block mode without concern about whether the echo facility is on or off. For block mode, press down the BLOCK MODE key and enter the input you desire. This mode is ideal when you plan a series of lengthy entries. For example, when you

make an error near the beginning of such an entry, you can skip the cursor back to correct this error without deleting any of the correct characters.

NOTE

You can dynamically switch from block mode to character mode simply by releasing the BLOCK MODE key. But, since MPE requires time to determine whether to echo the characters input, the first two characters echoed to the terminal by MPE may be lost if you enter your input faster than the currently-established input speed for your terminal. (At input speeds at or above 60 cps, this problem does not exist since such speeds exceed possible human typing rates.)

When you press the ENTER key in block mode, the terminal transmits the information from the current cursor position to the end of the line (or the end of the field, if the terminal is in format mode, as described in *HP 2640 Owner's Manual* and *HP 2644 Owner's Manual*).

To operate in block mode, the terminal must be strapped (strap no. 4) for Line/Field mode, as described in the above owner's manuals.

To illustrate block mode operation, consider the following example:

1. Suppose that you are using the terminal in character mode when you enter the following :FILE command:

```
:FILE DISCFILE; DEV=DISC2; REC=40,3,F,ASKII; DISC=1000,16,1
```

2. Because this command contains an error (the incorrect subparameter ASKII), an error message appears when you enter a carriage-return to transmit the command to the computer:

```
ERR 22,8
```

Error number (arrow pointing to 22)

*Erroneous parameter
(eighth parameter in the list)* (arrow pointing to 8)

3. To determine the meaning of Error Number 22, you enter any character other than a carriage-return. In response, the following message appears, followed by a prompt for your next command:

```
ILLEGAL PARAMETER  
:
```

Prompt (arrow pointing to :)

You now know that the eighth parameter in the parameter list is illegal. You realize now that you should have entered "ASCII" rather than "ASKII." To correct the single erroneous character (changing "K" to "C") in character mode, you would have to re-enter the entire :FILE command. You can, however, solve this problem much more efficiently by using Block Mode as follows.

4. Enter block (line) mode by latching down the *BLOCK MODE* key on the terminal. In this mode, no characters are transmitted until you press the *ENTER* key.
5. Using the cursor-positioning keys and the editing capabilities of the terminal, correct the erroneous parameter so that the :FILE command now appears as:

```
:FILE DISCFILE; DEV=DISC2; REC=40,3,F,ASCII; DISC=1000,16,1
```

6. Move the cursor to the beginning of the present line by entering a carriage-return. (This entry is not transmitted, since the terminal is still in block mode.)
7. Move the cursor one character to the right by pressing the right arrow (→) key. (You do this because you do not want the colon transmitted as part of your :FILE command; it has already been furnished by MPE as your prompt character.) Now the cursor is at the beginning of your corrected command.
8. Now transmit your corrected command by pressing the *ENTER* key. MPE will read and execute this command.
9. If you wish to return the terminal to character mode, simply unlatch the *BLOCK MODE* key.

NOTE

You may use the above procedure to correct any data to be read from your HP 2640 or 2644 Terminal or to re-enter any command or data that already appears on your screen.



RUNNING JOBS

SECTION

IV

MPE allows you to submit batch jobs via punched cards, magnetic or paper tape, or even a terminal. Unlike a session, a batch job is not interactive. Thus, you must enter the command stream into the system without prompting from MPE.

NOTE

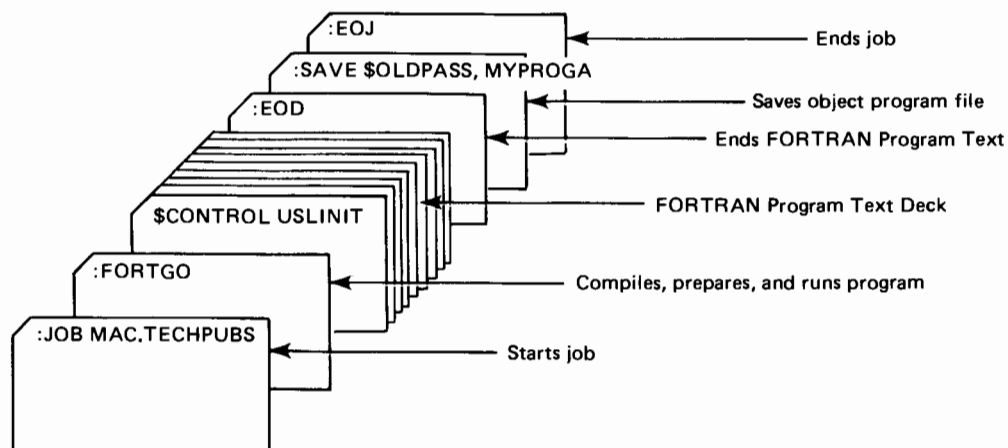
You can also initiate batch jobs while logged onto the computer in an interactive session. This technique, called *streaming jobs*, is discussed under *INTRODUCING JOBS WITHIN A SESSION*, later in this section. It allows you to enter your job from a terminal, card reader, magnetic tape unit, or disc file (prepared using the Editor). When the job is read, MPE spools it onto disc and processes it as an independent job completely separate from your session, allowing you to continue your session in the meantime.

The following example shows how to submit a typical batch job. In this job, the programmer is to perform these tasks:

1. Compile a source program written in FORTRAN language, prepare it into an object program, and execute the object program. This is the same FORTRAN program used in Section III to illustrate a session. However, in this job, the program is compiled from cards rather than disc.
2. Save the file in which the object program resides, for future use.

If you were running this job, you could proceed as shown below. (In this discussion, specific user actions and system responses are keyed to items in the job listing that appears in figure 4-1. User input is underlined.)

1. Organize the job deck as shown below:



2. Enter the job deck into the system through the card reader. MPE assigns the job a unique job number and schedules the job for access to the central processor. When the central processor is available, MPE verifies the user and account names entered in the :JOB command (figure 4-1, Item 1), used to initiate the job. At this point, MPE also verifies that you have the capability to run batch jobs (as defined by your account manager); that any limits established for central-processor and on-line connect time allowed your account and file group have not already been reached; and that the maximum number of jobs allowed on the system has not been reached.
3. If it verifies your access, MPE assigns a standard input file and a standard listing file for your job, and prints the job number and job-initiation information (Item 2) on the standard listing file. The standard input and listing files used in a job serve essentially the same functions as in a session. Unless a job is streamed, however, the standard input file/device is most often a card reader, and the standard listing file/device is most commonly a line printer; MPE copies all commands entered through the standard input file onto the standard listing file. There is always one standard input file/device and one standard listing file/device for each job.
4. The Command Interpreter encounters the :FORTGO command (Item 3) to compile, prepare, and execute your program. In response, the Command Interpreter first invokes the FORTRAN compiler, which reads the source text from the standard input file and compiles it. The text begins with a \$CONTROL USLINIT compiler subsystem command, used to initialize the file onto which the FORTRAN compiler writes its object code. (An :EOD command must be included to indicate the end of the source program input, but does not appear on the listing.) The compiler also outputs the following information to the standard listing file: a page-heading and identifying message (Item 4); a source-program listing (Item 5); a second page heading (Item 6); a verification that the program is error-free and successfully compiled, accompanied by compilation statistics (Item 7); and the message END OF COMPILE (Item 8). The compiler terminates, returning control to the Command Interpreter.
5. The Command Interpreter invokes the Segmenter, which prepares the compiled program into a temporary object program file (which you can later reference under the name \$OLDPASS), prints the message END OF PREPARE (Item 9), and returns control to the Command Interpreter.
6. The Command Interpreter now runs the object program, which produces its output (Item 10) on the standard list file, followed by the message END OF PROGRAM (Item 11). Again, control returns to the Command Interpreter.
7. The Command Interpreter executes the :SAVE command (Item 12), requesting the system to save the object program file (\$OLDPASS) under the permanent file name MYPROGA. This enables you to run the object program in future jobs and sessions.
8. The Command Interpreter encounters the :EOJ command (Item 13), which requests job termination. In response, the Command Interpreter closes all files that are open, and destroys all temporary files associated with your job; prints the central-processor and on-line time accumulated by the job (Item 14); adds the central-processor time to totals maintained at the account and group levels; prints the current date and time and the message END OF JOB (Item 15); and terminates the job. Now the Command Interpreter terminates.
9. When MPE begins printing the job output, it starts with a special header page (Item 16A) that identifies the job. When MPE terminates the job output, it concludes with a trailer page (Item 16B).

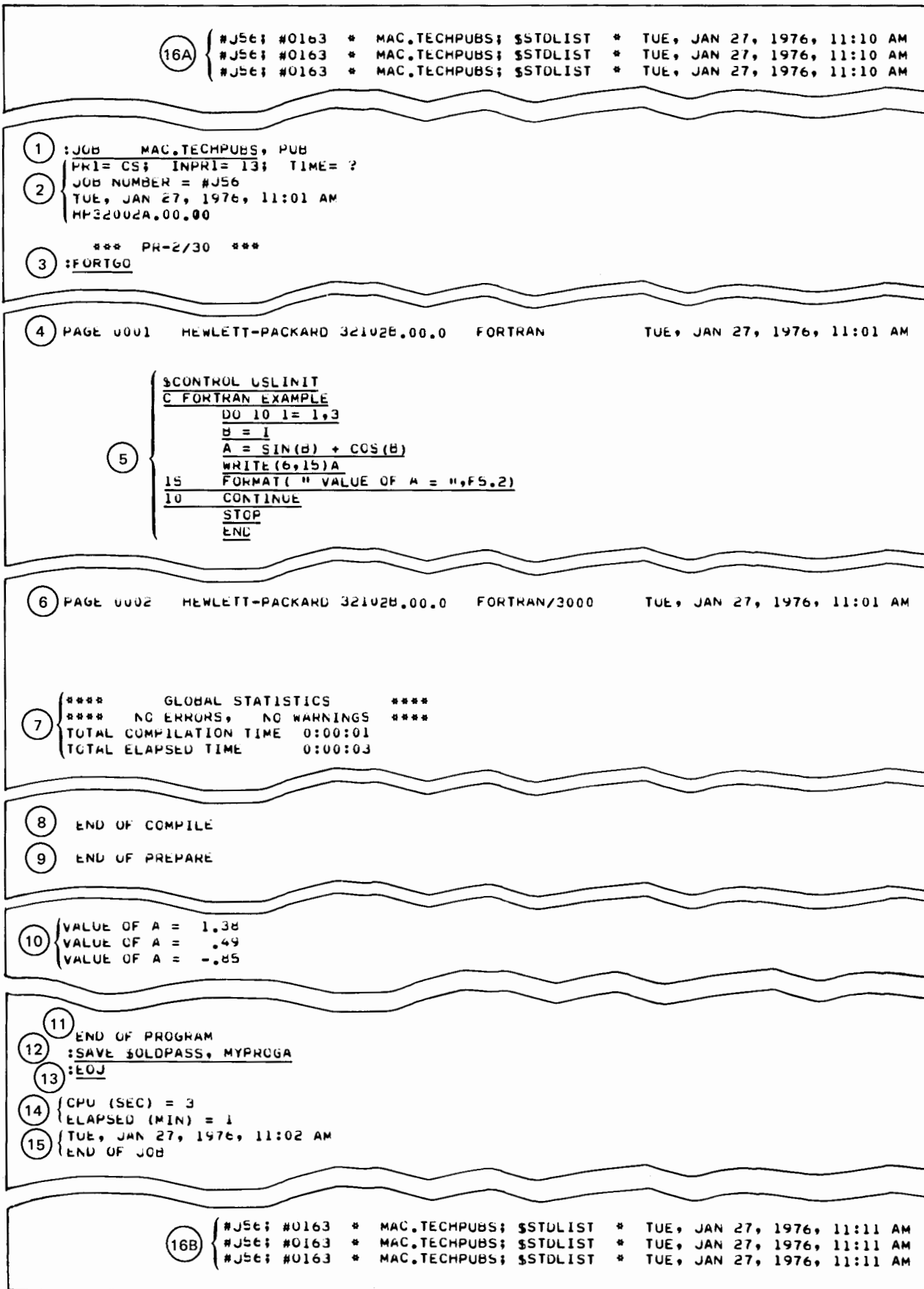


Figure 4-1. A Typical Job

INITIATING A JOB

You can enter a job through any device configured to accept jobs. If you use a card reader for this purpose, it is good practice to first check the card deck for proper order before loading it in the hopper. Basically, this means ensuring that the deck:

1. Begins with a :JOB command requesting job initiation.
2. Contains an :EOD command at the end of each program or group of data records (not immediately followed by :EOJ), acting as a delimiter.
3. Ends with an :EOJ command requesting job termination.

Beyond this, job structures may vary considerably from one application to another. Sample deck structures appear under *READING PROGRAMS AND DATA FROM STANDARD INPUT DEVICE* on page 4-11.

To enter a job from a card reader or magnetic or paper tape unit, load the cards or tape in the device and turn the device on-line. See table 3-1, page 3-6 for further details.

NOTE

You can also initiate one or more jobs from within a session, spooling them to disc for execution independent of the session. This is done through the :STREAM command, described on page 4-12.

The job is initiated by the :JOB command. Once this command is accepted, MPE reads and processes all succeeding commands sequentially until an :EOJ command is encountered or one of the conditions described under *PREMATURE TERMINATION OR SUSPENSION* (page 4-19) occurs. Much of the information you supply in the :JOB command is identical to that entered in the :HELLO command (Section III), and is only summarized in this section. Where differences exist, they are noted below.

INFORMATION ALWAYS REQUIRED

In the :JOB command as in the :HELLO command, you must always include your user name and account name, which you obtain from your Account Manager. If you omit either of these names or enter them incorrectly, MPE rejects your job and prints an error message on the standard listing device. If your job is accepted, however, MPE begins job processing. If the job is entered through a spooled input device, the job is copied to a disc file and initiated from that file rather than the originating device.

NOTE

The user and account name combination, sometimes used with an optional job name (described on page 4-6), constitutes a *fully-qualified job identity* — the minimum information required for the job initiation.

If the standard listing file is a line printer, MPE prints a special header page prior to listing the :JOB command, as shown below. (The Console Operator can disable the printing of this page via the console command = HEADOFF.)

<i>Unique job number assigned by MPE</i>	<i>User name</i>	<i>Account name</i>	<i>Standard List Device file name</i>	
#J8;	#059	* MAC.TECHPUBS;	\$STDLIST	* WED, OCT 8, 1975, 9:42 AM
#J8;	#059	* MAC.TECHPUBS;	\$STDLIST	* WED, OCT 8, 1975, 9:42 AM
#J8;	#059	* MAC.TECHPUBS;	\$STDLIST	* WED, OCT 8, 1975, 9:42 AM

Output file number assigned by MPE to Standard List Device

Date and time header record is actually printed.

Next, the :JOB command is listed, followed by the standard job-initiation display:

<i>Command identifier</i>	<i>User name</i>	<i>Account name</i>	
:JOB	MAC.TECHPUBS,	PUB	<i>Default :JOB command parameters supplied by MPE</i>
	PRI= DS;	INPRI= 13;	TIME= ?}
	JOB NUMBER =	#J8	<i>Unique job number assigned by MPE</i>
	WED, OCT 8,	1975,	9:41 AM <i>Current date and time</i>
	HP32002A.00.00		
	<i>MPE software part number</i>	<i>Version level</i>	<i>Update level</i>

The job number assigned by MPE always uniquely identifies your job to MPE and other users. MPE assigns such numbers in sequential order as jobs are accepted.

Sometimes, the job-acceptance information includes a message from the Console Operator following the standard display. When present, this is the same message output in the log-on information for sessions.

INFORMATION SOMETIMES REQUIRED

Under certain circumstances, you may need to furnish the following additional information in your :JOB command:

- File Group Name.
- User, Account, and/or Group Passwords.
- Terminal Type Code (for batch jobs initiated from terminals).

The cases in which this information is required and the rules for supplying it are the same as for the :HELLO command for sessions, except that:

1. If you omit a required password from the :JOB command, MPE rejects your access attempt without prompting you for the password.
2. When you enter the :JOB command through a device other than a terminal and the standard input device is different than the standard listing device, MPE never echoes passwords entered.
3. When the standard listing device is a line printer, and you do not specify a file group name, central-processor time limit, execution priority, and/or input priority in the :JOB command, the default values assigned by MPE for the omitted parameters appear on the job listing. (See preceding example.)

The following :JOB command, entered from a card reader, illustrates how to specify the group name MYGROUP and the group password MYPASS:

```
:JOB MAC.TECHPUBS, MYGROUP/MYPASS
```

OPTIONAL INFORMATION

For certain jobs, you may wish to specify additional information which is purely optional. Some of this information parallels information supplied in the :HELLO command for sessions, and is supplied according to the same rules. In this section, this information is discussed only with respect to its specific applications to batch jobs; further details appear in the description of the :HELLO command in Section III. This information includes:

- Job name qualifier.
- Central-processor time limit.
- Execution priority.
- Input priority.


Other information pertains to re-starting spooled jobs interrupted by system failure, and the output of spooled listing files. This information can be furnished only in the :JOB command, and is discussed in detail below.

JOB NAME (See also Section III). The *job name* parallels the *session name*, discussed in Section III. Thus, when several users are running under the same user names and account names, you can further qualify the identity of your job by including a job name, as follows, in your :JOB command:

Job name
↓
:JOB MYNAME, UNAME. ANAME
↑ ↑
User name *Account name*

CENTRAL PROCESSOR TIME LIMIT (See also Section III). This time limit functions in the same way as in a session. Thus, it allows you to limit the amount of central-processor time your job can use. This is particularly desirable in the early stages of program development, when you are testing a program for the first time, and wish to prevent the program from running indefinitely if it encounters an endless loop. Specify the limit in your :JOB command in terms of seconds, as follows:

```
:JOB MAC.TECHPUBS; TIME=50
```


*50-second central-processor
time limit*

When the time-limit expires, the entire job is aborted.

EXECUTION PRIORITY (See also Section III). The execution priority for jobs is used in the same way as that for sessions, and determines when the job is selected from among other jobs, sessions, or programs in the Ready List for access to the central processor. Normally, the default priority class at your installation is sufficient for most jobs. This default priority is normally DS, but the System Supervisor can change it to CS (via the :JOBPRI command, available only to himself). You may, however, wish to use a different execution priority. For instance, if the default is currently CS and your job requests a large compilation or other operation requiring extensive use of system resources, you may wish to assign your job the lower priority of DS. You can do this by entering:

```
:JOB MAC.TECHPUBS; PPI=DS
```

In this case, sessions and other jobs logging-on with CS priority access the central processor first. Your job runs with DS priority, accessing the central processor only when no other jobs, sessions or programs of the CS priority class are contending for it. When many sessions/jobs are running at CS priority, your job may require significant time for completion. But, when the session/job load is light, your job appears to access the central processor on nearly the same basis as sessions/jobs with CS priority.

INPUT PRIORITY (See also Section III). As with sessions, batch jobs are also assigned input priorities used to screen them against a job fence (if one is established by the Console Operator). The default input priority that applies to sessions also applies to jobs. Unlike sessions, however, jobs with input priorities less than or equal to the current job fence are merely delayed until the Operator raises their input priorities (with the =ALTJOB command) or lowers the job fence (via the =JOBFENCE command) — they are not rejected from the system. If you know the value of the current job fence, you can override the fence by specifying a higher input priority in the :JOB command. For instance, if the job fence is 8, you can run a job without delay by entering:

```
:JOB MAC.TECHPUBS; INPRI=9
```

If your job is delayed by the job fence, the input priority determines when the job leaves this waiting state and enters execution. For example, when several jobs are delayed and the operator lowers the job fence, those jobs whose input priorities now exceed the job fence will be initiated in order of descending input priority (10, 9, 8, . . . and so forth). When two or more jobs have the same input priority, the jobs

are executed in order of their entry into the system. Jobs delayed because they were awaiting the availability of resources (such as list devices) are executed as soon as those resources become available, ahead of those jobs delayed solely because of input priority.

As in a session, if you have the System Manager or Supervisor user capability, you can override *any* job fence value by including the HIPRI parameter in your :JOB command:

```
:JOB MAC.TECHPUBS; HIPRI
```

RE-STARTING INTERRUPTED JOBS. When a system termination (whether normal or abnormal) occurs, all jobs and sessions executing at the time are also terminated and must later be re-initiated from the beginning. However, if you initiate a batch job on a spooled input device, you can designate the job as *re-startable* in the following way:

```
:JOB MAC.TECHPUBS; RESTART
```

↖
Designates re-startable job

If this job is interrupted by a system termination, and the system is subsequently re-initiated with the WARMSTART option, MPE then automatically re-starts the job at the point where the interruption occurred.

NOTE

Because a *power failure* generally has a temporary rather than a permanent impact on sessions and jobs, as discussed on page 3-22 and 4-20, it is regarded as a system *suspension* rather than a system *termination*. Typically, when power is restored following failure, all batch jobs continue at the points where they were interrupted *regardless* of whether they were designated as re-startable or not in the :JOB command.

CONTROLLING JOB OUTPUT. Normally, as your job is executed, the following information is written on the standard listing device: MPE commands, any printed responses to those commands, source program listings, and object program output not specifically directed to other devices. In some cases, however, you may wish this information written to another device. For instance, when entering a job from a card reader with a certain printer configured as the standard listing device, you may wish to select a different printer for your output. This may be the case where the primary printer is temporarily unavailable because of a malfunction or heavy system printing load. In this case, you can re-direct output to a secondary printer by referencing the unique logical device number of the desired printer in your job command, as follows:

```
:JOB MAC.TECHPUBS; OUTCLASS=12
```

↖
Logical device number of printer

NOTE

You can also select a specific device by using its device class name, if that device is the only one configured with that class name. For further information about logical device numbers and device class names, see Section VI.

MPE allows you to control the output of spooled files produced by a job by assigning output priorities to those files. These priorities may range from 1 (lowest) to 13 (highest). The actual printing of the files is managed by the Console Operator, who establishes an *outfence* (with the =OUTFENCE command) that relates to output file selection in the same way that the *jobfence* relates to job selection. That is, spooled output files with priorities lower than or equal to the outfence are not printed until the outfence is lowered or the priorities are raised. As an example, if you are running jobs that spool output to disc overnight or at any other time that the computer is unattended, you can enter your jobs with output priorities less than or equal to the current outfence. Then, you simply request the Operator to lower the outfence when he returns to the computer. The jobs are executed as soon as possible, but their output is delayed until the outfence is lowered. This technique is valuable as a safeguard against problems arising from the printer jamming while the system is unattended. In the following :JOB command, output priority 9 is requested:

```
:JOB MAC.TECHPUBS; OUTCLASS=12, 9
```

Logical device number of printer *Output priority number*

If you should require more than one copy of your job listing output, and this listing is directed to a spooled output device, you can request these copies by supplying another :JOB command parameter. For example, to request three copies, enter the last parameter shown below:

```
:JOB MAC.TECHPUBS; OUTCLASS=12, 9, 3
```

Logical device number of printer
Output priority number *Request for three copies of listing*

This technique is highly valuable when several copies of listings for large programs requiring extensive execution time are needed, for it allows you to obtain as many copies as you desire while running the program only once.

NOTE

The above spooling features apply only when the job listing is directed to a spooled line printer or card punch. They can also be requested for individual files through the :FILE command (Section VI) or programmatically through the FOPEN intrinsic.

ERRORS DURING JOB ENTRY

If the connect time, central processor time, or file space limit allotted to your account or group has been exceeded by previous sessions and jobs, your job is rejected upon submission. You must then ask the System Manager to adjust or clear these limits and enter your job again. If these limits expire during a job, however, the job is allowed to continue; the next job or session attempted under this account or group is rejected.

If you enter your job through a terminal, you must complete the :JOB command within a time limit specified during system configuration (typically, two minutes); if you fail to do so, MPE disconnects your terminal and you must once more establish contact with the system.

If you enter an illegitimate :JOB command (with an incorrect user, account, or group name or a bad password, for instance), your job is rejected. A job is also rejected when you try to enter it without having the user capability to run jobs; when you use an incorrect delimiter in your :JOB command parameter list; when you specify an improper central processor time limit; or when you request an improper priority.

Errors in the :JOB command are handled in the same way as errors in the :HELLO command, discussed on page 3-13, except that you cannot interactively recover from such errors.

TERMINATING A JOB

To denote the end of a job, use the :EOJ command. In response to this command, MPE outputs a standard job-termination display and terminates the job. This display shows the central-processor time (in seconds) and time elapsed since the beginning of the job (rounded upward to the nearest minute), the current date and time, and the message END OF JOB, as follows:

```
:EOJ
CPU (SEC) = 14
ELAPSED = 2
WED, OCT 8, 1975, 9:42 AM
END OF JOB
```

MPE also adds the central processor time and file space used by your job to the resource-usage counters maintained for your log-on account and group.

If you omit the :EOJ command from a job, the next :JOB command terminates the current job and starts a new one. (The end of the first job is indicated by the standard end-of-job display, and the beginning of the next job is denoted by the job-initiation display, in the normal way.)

If the job listing file is directed to a line printer, MPE prints a job trailer page after the job listing. This page contains the same information as the job header page, as follows. (The Console Operator can disable printing of this page via the =HEADOFF console command.)

```
#J8; #059 * MAC.TECHPUBS; $STDLIST * WED, OCT 8, 1975, 9:42 AM
#J8; #059 * MAC.TECHPUBS; $STDLIST * WED, OCT 8, 1975, 9:42 AM
#J8; #059 * MAC.TECHPUBS; $STDLIST * WED, OCT 8, 1975, 9:42 AM
```

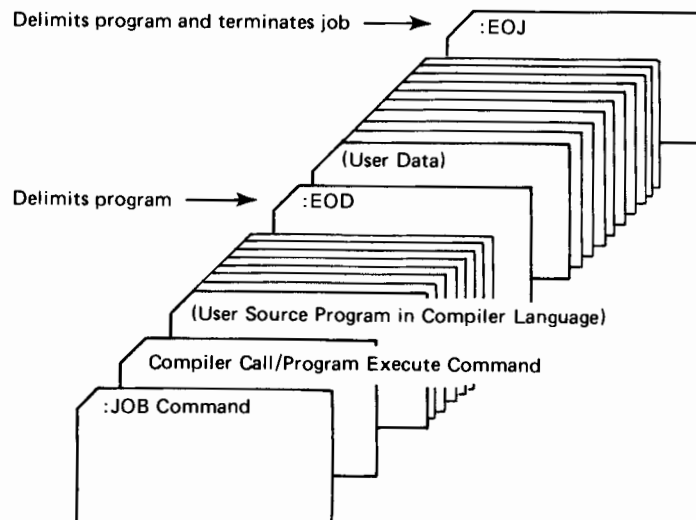
READING PROGRAMS AND DATA FROM STANDARD INPUT DEVICE

In a job as in a session, you can enter source programs and data as well as MPE commands from the standard input device. When you do this, you should logically delimit both the programs and data with an :EOD command, except where the program or data is immediately followed by the end-of-job (:EOJ) command; where :EOJ appears, it both terminates the job *and* delimits the program or data. (Some compilers provide source statements to terminate compilations, while others do not. Because it is easy to accidentally omit such statements when they are required, and because you *must* use an :EOD command to delimit compilation when no source statement is available for this purpose, it is good practice to enter :EOD after every source program not immediately followed by :EOJ. This procedure ensures proper delimiting of your source record input. It is *essential* if your standard input device has been opened with the file name \$STDINX (Section VI) in a batch job.)

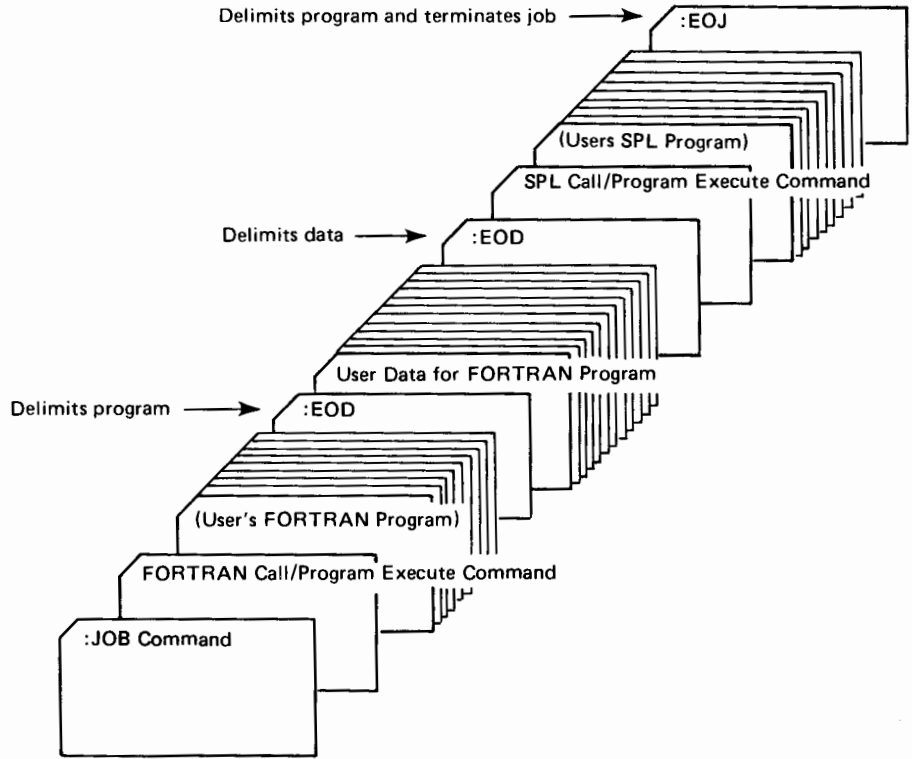
NOTE

In addition to performing their primary functions, *all* MPE commands read during a batch job also delimit data on the standard input device. But most programmers use :EOD for delimiting data because this command requests no other MPE operations.

As an example, the job shown below includes a source program and data to be processed during its execution. Note that the :EOD command follows the program but is not required after the data (because the data is delimited by :EOJ).



As another example, the following job contains: a FORTRAN program to be compiled and executed; data for that program; and an SPL program to be compiled and executed with data from a disc file. This job requires two :EOD commands, as shown:



More information on the :EOD command appears on page 3-19.

INTRODUCING JOBS WITHIN A SESSION

In addition to entering batch jobs in the usual way, you can also initiate them while logged onto the computer in an interactive session. This technique is called *streaming jobs*. When you use streaming, MPE allows you to enter the records making up your job from your terminal or to read these records from a card or tape file. Alternatively, you can read these records from a disc file previously prepared via the Editor. When the job is read, MPE spools it onto a disc file, assigns it a job number, and processes it independently as a completely-separate entity from your session. In the meantime, MPE allows you to continue with your session. From this point on during the session, you can check the status of the batch job by entering the :SHOWJOB command, discussed in Section VIII.

NOTE

You can only initiate jobs in this way if the Console Operator has enabled the *MPE Streaming Facility* by entering the console command =STREAMS. This command also specifies a *streaming device*, which (to MPE) appears to be the source of your job input regardless of what device you actually use for this input. Thus, the job number assigned by MPE and the listing generated by the job appear on the standard listing device that corresponds to the streaming device, rather than on your terminal.

In addition to jobs, you may also stream data to disc if the input device used is configured to accept the :DATA command (Section VI).

You would use streaming, for instance, if you were logged-on in a session and wished to compile, prepare, and execute the FORTRAN program MYPROG independently from your session without logging off. You could do this by streaming a job that performs these operations as follows:

1. During the session, request MPE to read the commands making up the job from your terminal by entering:

:STREAM

2. In response, MPE prompts you for each command by printing a greater-than (>) sign. After each prompt, enter the command desired, terminating each record with a carriage-return. You cannot, however, begin the command with the colon (:) normally used to identify MPE commands; instead, you must use a substitute character as the command identifier. In this case, the default substitute character, which is an exclamation point (!), is used. Following the !EOJ command that terminates the job, MPE prints a job number on your terminal and schedules the job for execution:

```
      .  
      .  
      .  
>!JOB MAC.TECHPUBS      Initiates batch job.  
>!FORTGO MYPROG        Compiles, prepares, executes program.  
>!EOJ                  Terminates batch job.  
#J6                    Job number assigned by MPE.  
      .  
      .  
      .  
      .
```

3. Terminate the job input by entering a colon (:). In response, MPE terminates prompting for batch job input and instead prompts you for another MPE command:

```
>:  
:  
      Denotes end of batch job input.  
      Prompt for next MPE command.
```

In some cases, you may wish to use a device other than your standard input device for the batch job input. For instance, you could use the Editor to enter the job into a disc file and then stream the job from that file, thus bypassing punched cards as the most convenient medium for batch job input. In such a case, however, you must name the input file as a parameter in the :STREAM command:

:STREAM DISCFILE

↑
Name of file on disc that contains job input.

If you use a character other than an exclamation point as the substitute command identifier in your job input, you must identify that character in the :STREAM command. Because you enter this character as the second positional parameter in this command, you must always precede it with a delimiting comma, even when you omit the input file name (the first parameter):

```

:STREAM , *
>*JOB MAC.TECHPUBS
>*FORTGO MYPROG
>*EOJ
#J74
>:
:

```

Delimiting comma

Asterisk used as substitute command identifier

NOTE

If your job input file contains subsystem commands, such as commands directed to the Editor, do *not* enter *any* command identifier at the beginning of these commands. For instance, when using the Editor, enter the subsystem commands as follows:

<pre> :STREAM , # >#JOB MAC.TECHPUBS >#EDITOR >TEXT MYFILE >ADD . . . >KEEP MYFILE, UNNUMBERED >EXIT >#EOJ #J76 >: : </pre>	<p><i>Initiates streaming.</i></p> <p><i>Initiates job.</i></p> <p><i>Invokes Editor.</i></p> <p style="font-size: 2em;">}</p> <p><i>Subsystem commands (without # as subsystem command identifier).</i></p> <p><i>Terminates job.</i></p> <p><i>Job number.</i></p> <p><i>Terminates job input.</i></p> <p><i>Prompt for new MPE command.</i></p>
---	--

(In the above example, MYFILE is the name of the file on which your edited text is written; you must always name such a text file in the Editor's TEXT statement, or the operation will fail.)

You cannot use the :STREAM command to spool job records beginning with a colon into the system from a device configured to accept jobs, sessions, or data entered via the :DATA command. But, suppose you have a job whose records begin with a colon, already punched on cards; you wish to spool this job from a job-accepting card reader during a session, without converting the command identifier on each record from a colon to some other character. You could do this as follows:

1. Place the job deck in the card reader and request the Console Operator to disallow sessions, jobs, and data (entered via the :DATA command) on this device. The Operator does this by entering the = REFUSE console command.

2. Copy the job to disc, using the MPE FCOPY facility.
3. Spool the job into the system as follows:

```
:STREAM DISCFILE
      ↑
      |
      | Name of input file on disc
```

4. Request the Console Operator to return the card reader to its previous session/job/data-accepting status.

Normally, by default, a streamed job is executed under the DS priority class (or the CS priority class, if the System Supervisor has assigned this class as the default). Should you wish to assign the job another priority, you can do so in the JOB command as follows:

```
:STREAM
>!JOB MAC.TECHPUBS; PRI=CS
.
.
.
      ↑
      |
      | Alternate priority class
```

If you wish the job listing to appear on a device other than the standard listing device associated with the streaming device, you can specify this other device in the JOB command in this way:

```
:STREAM
>!JOB MAC.TECHPUBS; OUTCLASS=12
      ↑
      |
      | Logical device number of alternate device
```

Do not allow any record in the job input file to exceed 255 characters; otherwise, an error results. When you enter a command in the input file that exceeds the physical record width permitted by the input device — for example, a command longer than 80 characters on a card reader — use the MPE continuation character (&) at the end of each line of the command except the last.

When you interrupt execution of the :STREAM command by pressing the BREAK key on the terminal, this command and streamed jobs not yet initiated are aborted, and incompletely-spoiled disc space is returned to the system.

If you make an error while entering the JOB command, you receive an error message on your job listing device. The Console Operator, however, receives no indication of either the job or the error.

NOTE

If you enter your job input file on cards punched on an IBM 029 keypunch, remember that certain keyboard characters do not correspond with their ASCII (Hollerith) representations. In particular, striking the exclamation point (!) key actually punches the ASCII right-bracket (]). This means that when you punch your input file on this keypunch and use an exclamation point as a substitute command identifier, you must *explicitly* specify the command identifier in your :STREAM command, as follows:

1. If you also enter the :STREAM command on a card punched on the IBM 029 keypunch, specify:

```
:STREAM , !
```

Command identifier, interpreted as] in this command

2. If you enter the :STREAM command through another medium, enter:

```
:STREAM CARDFL, ]
```

Command identifier, matching identifier used in card file

File name of input file on cards

If you do not follow the above rules, the command identifier specified in :STREAM will not match the one actually used in the job input file, and an error results.

INSERTING COMMENTS IN A JOB

Occasionally, you may want to include comments or notes in job listings produced on hard-copy devices to create headings or explain the purpose of commands or logic used. You can do this by entering :COMMENT followed by text consisting of any ASCII characters you desire. If necessary, you can continue the text onto more than one line by entering the MPE continuation character (&) at the end of each line to be continued. When MPE prints the job listing, it includes the :COMMENT command verbatim — the command name as well as the text entered. Comments are most useful in batch jobs, but you can employ them in sessions as well. In the example below, a comment used as a job heading appears:

```
:JOB MAC.TECHPUBS, PUB
PRI= DS; INPRI= 13; TIME= ?
JOB NUMBER = #J8
WED, OCT 8, 1975, 9:41 AM
HP32002A.00.00

→ :COMMENT THIS IS A SAMPLE JOB.
:FOR TGO

:
```

When you wish to insert comments within listings furnished by a compiler or subsystem, you can usually do this through commands or fields provided by the subsystem. See the manuals covering the subsystems for further details.

EFFECTS OF SPOOLING

When spooling is in effect, it relieves contention between your batch job and other jobs and sessions for a particular device by buffering job input/output to disc. Thus, input entered on the originating peripheral device is copied to a *spooled devicefile* on disc before it is processed, freeing the device for use by other jobs and sessions. Output destined for a particular output device is written to a spooled devicefile, allowing your job to continue without waiting for the destination device to become available; when the device is free and your job is complete, MPE copies the output from the spooled devicefile to the destination device.

Input files can be spooled from card reader or magnetic tape units configured to accept jobs, sessions, or data. When jobs are input from these devices and spooling is in effect, the entire job is copied to a spooled devicefile on disc and executed by MPE from that file. (As noted earlier, several jobs can be spooled by using the `:STREAM` command.) Thus, the number of jobs that can execute concurrently is not limited to the number of job-accepting devices connected to the system, as it would be without spooling. (Without spooling, for example, you would load your job into the card reader and wait until the job had completed execution and freed the card reader before you could use that device for another job.) Additionally, input files used by your job can also be spooled onto disc where they can be read by the job, freeing the originating device for use by other jobs. When a job or other input file is being copied to a spooled devicefile, it is in the `ACTIVE` state; files in this state cannot be accessed by jobs or sessions. When the spooling is complete and the entire input file resides on disc, the file enters the `READY` state, awaiting access by user or system programs. When access to the file begins, the file enters the `OPENED` state.

Output files can be spooled to line printers, card punches, and plotters. Output spooling for batch jobs operates in the same way as for sessions, with the exception that you can specify in the `:JOB` command:

- *The destination device for a spooled job-listing file.* This feature enables you to select a specific line printer or card punch for your listing output.
- *The output priority for a spooled job-listing file.* Different spooled files, all destined for the same output device, are printed or punched in order of their priority; those with the same output priority are copied on a first-in, first-out basis. If you select an output priority less than or equal to the current outfence, output of the job listing file is deferred until the Console Operator lowers the outfence (with the `=OUTFENCE` command) or raises the output priority of the file (via the `=ALTFILE` command).
- *The number of copies of the job-listing file to be printed.* Each time a copy is complete, the spooled devicefile is transferred from the `ACTIVE` to the `READY` state and is rescheduled for output. When the last copy is made, the spooled devicefile is deleted.

Spooling has very little apparent impact on your job. In fact, you cannot generally tell whether you are accessing a spooled devicefile on disc or an actual input or output device. Each time you access a spooled file, you handle this access programmatically as though conducting input/output on the originating or destination device; MPE automatically directs your access to the spooled devicefile on disc.

NOTE

In a spooling environment, if an output devicefile runs out of available disc space, no new spooling will occur and the message (SPACED OUT) is printed on the trailer page of the output listing. Should any other error occur, the message (INCOMPLETE) appears on the trailer page. See your Console Operator for corrective action.

Although certain devices can be spooled automatically when the system is cold-loaded, the Console Operator can initiate spooling on other devices not currently being used by a job or session (through the =SPOOL console command). The Operator can change the output priority or destination of a spooled output file. He can also change the number of copies to be produced for a READY or OPENED spooled output file. He can place the spooling program (*spooler*) in the WAIT state so that it completes ACTIVE file spooling in progress but does not initiate spooling of other pending files. He can also delete or defer the current ACTIVE file and suspend spooling immediately.

When the system terminates (whether normally or abnormally) and the Operator subsequently restarts it via the WARMSTART operation, MPE begins file recovery procedures. (WARMSTART is explained in the manual *Console Operator's Guide*.) For spooled input files, the following action takes place and the job fence is automatically reset to 14, deferring all jobs and sessions except those with HIPRI input priority:

SPOOLED INPUT FILES

File State	Action
ACTIVE	Deletes this file; for recovery, you must re-initiate job.
READY	Retains this file in READY state. (This includes jobs placed in the WAIT category and unstarted jobs whose spooling is complete.)
OPENED	If this file contains a job not designated as <i>restartable</i> , it is deleted. If this file contains either a <i>restartable</i> job or no job at all, the file is returned to the READY state and completely recovered; a restartable job is re-scheduled for completion.

NOTE

The file states can be displayed by entering the :SHOWIN or :SHOWJOB commands.

For spooled output files, the following action occurs, and the outfence is set to 14, deferring output of all spooled files:

SPOOLED OUTPUT FILES

File State	Action
OPENED	Places file in READY state, with output to file incomplete. MPE prints or punches as much of the output as possible, but since output to the spooled devicefile was not completed, printing/punching terminates with the message (INCOMPLETE) on the trailer page. In addition, the message SPOOFL I/O ERROR may appear on the console in which case the file is deferred and may be deleted by the Console Operator.
READY	Retains file in READY state.
ACTIVE	Returns file to READY state. All extents are saved and any remaining copies are produced when file returns to ACTIVE state.

NOTE

The file states can be displayed by entering the :SHOWOUT command.

For further information about spooling, see Sections III and VIII.



PREMATURE JOB TERMINATION OR SUSPENSION

Under certain circumstances, MPE terminates or suspends your job before it encounters the :EOJ command. Specifically, MPE immediately terminates the job if:

1. An error occurs in the interpretation or execution of an MPE command.
2. A subsystem error occurs.
3. A program run by the job is aborted.
4. The Console Operator issues the = ABORTJOB command (to terminate the job), the = LOGOFF command (to abort all executing jobs and sessions and prevent further logging-on), or the = SHUTDOWN command (to shut down the system).
5. MPE encounters a second :JOB command (to begin another batch job from your standard input device) or a :DATA command (to establish the standard input device as a data file that can be acquired by other jobs and sessions). These commands always result in job termination.
6. MPE encounters a hardware end-of-file (or the :EOF: command, used to simulate this) on the standard input device.
7. The job exceeds the central-processor time limit established by the *cpusecs* parameter of the :JOB command (or its default value).

The first three of the above cases place the job in the *error state*, which normally results in job termination. But if you anticipate an error as a result of a specific MPE command, you can suspend this termination by preceding that command with :CONTINUE. The :CONTINUE command permits your job to continue even though the following command results in an error (in which case the error-state indication is re-set). For instance, suppose you submit a job to execute three programs (with the :RUN command), but anticipate a probable terminating error in the first program. To continue the job and execute the second and third programs, use the :CONTINUE command as follows:

```
→ :JOB JNAME,UNAM.ACCTNAM
   :CONTINUE
   :RUN PROG1
   :RUN PROG2
   :RUN PROG3
   :EOJ
```

NOTE

The :CONTINUE command operates only on the command which immediately follows it; therefore, if an error occurs in PROG2 in the above example, the job is aborted.

Although the Console Operator can abort your job, you as a user cannot issue a command within a session that aborts a job within the system.

When a system termination (whether normal or abnormal occurs), all batch jobs executing at the time are terminated and must be re-initiated from the beginning (unless they are spooled jobs designated as *re-startable* in the :JOB command); re-startable jobs are temporarily suspended until the Console Operator re-initiates the system with the WARMSTART option.

NOTE

Because a *power failure* generally has a temporary rather than a permanent impact on jobs (and sessions), as described below, it is regarded as a system *suspension* rather than a system *termination*.

When a system power failure occurs, MPE preserves the operating environment prior to complete loss of power. When power returns, normal system operations usually resume and batch jobs continue at the points where they were interrupted. There are, however, some input/output errors from which the system cannot recover; for instance, any input/output from a device other than a disc or terminal is aborted.

NOTE

If you are entering input from a terminal when power fails, you may be uncertain which characters already entered in the current record were actually received. In this case, enter the CONTROL-X (X^c) character to clear the input buffer; then re-enter the current record.

When your job directs output to a spooled output device, certain commands issued by the Console Operator may prevent or suspend output of those files even though the job itself is not terminated or interrupted. These commands are: =DELETE (to delete READY devicefiles); =SPOOL *1dn*,STOP,DELETE (to terminate output spooling immediately and delete the ACTIVE spooled devicefile); =SPOOL *1dn*,WAIT (to suspend spooling after completing the ACTIVE devicefile); =SPOOL *1dn*,DEFER (to change the output priority of the ACTIVE spooled devicefile, and return it to the READY state); =SPOOL *1dn*,DELETE (to delete the currently-ACTIVE devicefile); and =SPOOL *1dn*,SHUTQ (to prevent creation of new spooled devicefiles for a device). The =SPOOL *1dn*,SHUTQ command could prevent jobs from opening files on a spooled output device; it could also keep jobs in the WAIT state until the Operator issues the =SPOOL *1dn*,OPENQ command (to allow creation of new spooled devicefiles) or the output device becomes available to the system.

ERROR-HANDLING

If a command in a batch job contains an error (and you do not precede the erroneous command with :CONTINUE), MPE suppresses execution of that command, prints an error number and message on the job listing device, follows this with the *PREMATURE JOB TERMINATION* message, ignores all subsequent commands, and aborts the job. When the error can be directly attributed to a particular command parameter, the error number is followed by a number that denotes the sequential location of the bad parameter in the parameter list. For example, the following :RUN command produces the error number 108 and the corresponding message *NON-EXISTENT FILE*. The number 1 implies that the error resulted from the first (and in this case the only) parameter in the command, which names a file that does not exist:

```
      .  
      .  
      .  
:RUN MYPROGY  
ERR 108.1 NON-EXISTENT FILE  
PREMATURE JOB TERMINATION  
      .  
      .  
      .
```

When unusual but non-fatal conditions arise during interpretation or execution of a command, MPE executes the command (if possible), issues a warning message following the command, and allows the job to continue.

A full discussion of Command Interpreter error and warning messages appears in Section X.

DETERMINING SESSION/JOB STATUS AND RESOURCE USAGE

SECTION

V

While a session or job has access to the system, it enters certain processing states and uses various resources that are charged to your log-on account and group. Since you may need to be aware of them from time to time, the MPE processing states and resource accounting methods are explained in this section.

HOW MPE HANDLES SESSIONS AND JOBS

When you successfully initiate an interactive session, that session immediately acquires access to the central processor and enters the Executing (EXEC) processing state. As long as you remain logged on, your session remains in that state. But when you enter the :BYE command to log-off, the session enters the DONE state, where it remains until MPE completes its log-off message and terminates communication with you. At this point, the session is over; spooled output produced by the session, however, may still exist in the system until its required destination device is available.

The life cycle of a batch job may be more complex than that of a session. If you input the job from a spooled device, the job enters the Introduction (INTRO) state and remains there while MPE spools it to disc. If the job cannot be executed when spooling is complete, MPE places the job in the WAIT state. This occurs when the job requires resources (such as file space or a listing device) that are not immediately available; for example, a job requiring an unspooled line printer presently used by another job enters the WAIT state until that printer is free. A job also enters the WAIT state if the maximum number of jobs allowed to execute concurrently has been attained; it leaves this state when one of the running jobs terminates. (This job limit is established by the System Supervisor during system configuration and may be subsequently raised or lowered by the Console Operator through the = LIMIT command.) In addition, a job enters the WAIT state if its input priority does not exceed the current job fence; in this case, the job is also said to be DEFERRED and is allowed to execute when the Operator lowers the job fence (by entering the = JOBFENCE command) or raises the input priority of the job (via the = ALTJOB command). When a job leaves the WAIT state and accesses the central processor, or when it is initiated from an unspooled device, the job enters the EXEC state. Normally, the job remains in this state until terminated, even though it may repeatedly relinquish and regain central-processor access during this period. However, the Console Operator may temporarily suspend the job, placing it in the SUSP state, by entering the = BREAKJOB command; the job returns to the EXEC state when the Operator enters the = RESUMEJOB command. Finally, when the job is complete and MPE begins termination operations for it, the job enters the DONE state. The job remains in this state until it is completely removed from the system. (Beyond this point, however, spooled output produced by the job may still exist in the system.)

Occasionally you may need to determine the status of sessions and jobs currently logged-on. For example, you may plan to run a lengthy session requiring fast response time from the system, and wish to determine that the current session/job load is not too great to preclude this. You can find out the number of sessions and jobs in each processing state, plus the current jobfence and session/job limits, by entering the :SHOWJOB command as follows:

Total number of sessions and jobs

:SHOWJOB STATUS

6 JOBS:

0 INTRO

0 WAIT; INCL 0 DEFERRED

6 EXEC; INCL 6 SESSIONS

0 SUSP

} Number of sessions and jobs in each state

JOBFENCE= 0; JLIMIT= 3; SLIMIT= 16

Current jobfence

Current job limit

Current session limit

With the :SHOWJOB command, you can also keep track of individual spooled and streamed jobs entered in the system. In fact, this is the *only* way in which you can do so. Suppose, for example, that you enter two batch jobs identified by the job names JOB1 and JOB2, under the user name MAC and the account name TECHPUBS. If you do not receive the output from these jobs in a reasonable length of time, you can determine their status by logging-on and entering:

Request for status of all sessions/jobs under user name MAC, account name TECHPUBS

Processing state

Logical device number or device class name of standard listing device

:SHOWJOB JOB=0, MAC.TECHPUBS

JOBNUM	STATE	IPRI	JIN	JLIST	INTRODUCED	JOB NAME
#S90	EXEC		41	41	FRI 7:54A	SESS1,MAC.TECHPUBS
#J16	WAIT	D 5	5S	LP	FRI 8:10A	JOB1,MAC.TECHPUBS
#J17	WAIT	D 5	5S	LP	FRI 8:10A	JOB2,MAC.TECHPUBS

Day and time of introduction

Session/job identifier

3 JOBS (DISPLAYED):

0 INTRO

2 WAIT; INCL 2 DEFERRED

1 EXEC; INCL 1 SESSIONS

0 SUSP

JOBFENCE= 6; JLIMIT= 3; SLIMIT= 16

Session/job number

D=Deferred job

S=Spooled input device

Logical device number or device class name of standard input device

Input priority (for jobs not in execution)

The above command produces a status report for all sessions and jobs running under the identifier MAC.TECHPUBS. This report reveals that both JOB1 and JOB2 have been placed in the WAIT, DEFERRED state because their input priorities (both 5) are lower than the current job fence (6). The

session under which you are logged-on (SESS1), of course, is in the EXEC state. When a session or job is executing, no input priority is listed for that session/job on the status report; for this reason, the IPRI entry for SESS1 is blank. When a session or job is in the INTRO or WAIT state with HIPRI input priority, this is noted in the report as *HI*; the above reports lists no jobs of this priority.

NOTE

The notation @ in a command parameter indicates *all*. Thus, @,USER.ACCTA means all sessions/jobs running under the user name USER and the account ACCTA; @,@.ACCTA means all sessions/jobs under *all* user names on Account ACCTA.

Because you are particularly concerned about receiving output from JOB2, you return to the terminal 15 minutes later to check the status of that job. Since you now know the unique job number assigned to JOB2 by MPE, you can enter:

```

: SHOWJOB #J17
JOBNUM  STATE IPRI  JIN  JLIST    INTRODUCED  JOB NAME
#J17    EXEC           5R LP      FRI  8:10A  JOB2,MAC.TECHPUBS
JOBFENCE= 0; JLIMIT= 3; SLIMIT= 16

```

Job number (arrow pointing to #J17)

R = Job is being executed with RESTART option (arrow pointing to 5R LP)

The above report tells you that the Console Operator has lowered the job fence to zero, allowing JOB2 to execute. Two minutes later, you enter the :SHOWJOB command again:

```

: SHOWJOB #J17
NO SUCH JOB(S)
JOBFENCE= 0; JLIMIT= 3; SLIMIT= 16

```

The above report indicates that JOB2 has completed execution, passed through the DONE state, and is no longer in the system.

If you wish to determine the status of a single session or job but do not know its session or job number, you can enter its session or job identity in the :SHOWJOB command, as follows:

Fully-qualified job identity

```

: SHOWJOB JOB=JOB1, MAC.TECHPUBS
JOBNUM  STATE IPRI  JIN  JLIST    INTRODUCED  JOB NAME
#J14    WAIT   D 5    5S LP      FRI  7:56A  JOB1,MAC.TECHPUBS
JOBFENCE= 6; JLIMIT= 3; SLIMIT= 16

```

To obtain a report on all sessions and jobs under your (or any other) account, enter :SHOWJOB in the format shown below:

Account name
↓

```
:SHOWJOB JOB=@, @.ALANG
```

JOBNUM	STATE	IPRI	JIN	JLIST	INTRODUCED	JOB NAME
#S702	EXEC		25	25	TUE 1:21P	SPL.ALANG
#S704	EXEC		34	34	TUE 1:24P	SPLT.ALANG
#S713	EXEC		29	29	TUE 1:38P	DL, SPL.ALANG
#J391	EXEC		6R	FASTLP	TUE 2:00P	BARRETT, SPL.ALANG
#J392	WAIT:1	13	6S	FASTLP	TUE 2:05P	SPLT.ALANG

5 JOBS (DISPLAYED):
 0 INTRO
 1 WAIT; INCL 0 DEFERRED
 4 EXEC; INCL 3 SESSIONS
 0 SUSP
 JOBFENCE= 2; JLIMIT= 1; SLIMIT= 16

If you wish to determine the status of all batch jobs in the system, enter:

```
:SHOWJOB JOB=@J
```

JOBNUM	STATE	IPRI	JIN	JLIST	INTRODUCED	JOB NAME
#J391	EXEC		6R	FASTLP	TUE 2:00P	BARRETT, SPL.ALANG
#J392	WAIT:1	13	6S	FASTLP	TUE 2:05P	SPLT.ALANG

2 JOBS (DISPLAYED):
 0 INTRO
 1 WAIT; INCL 0 DEFERRED
 1 EXEC; INCL 0 SESSIONS
 0 SUSP
 JOBFENCE= 2; JLIMIT= 1; SLIMIT= 16

When you wish to determine which sessions/jobs under your user name and account are in a particular state (such as WAIT), you can request a report on them in this way:

User name *Account name* *State*
↓ ↓ ↓

```
:SHOWJOB JOB=@, SPLT.ALANG; WAIT
```

JOBNUM	STATE	IPRI	JIN	JLIST	INTRODUCED	JOB NAME
#J392	WAIT:2	13	6S	FASTLP	TUE 2:05P	SPLT.ALANG

JOBFENCE= 2; JLIMIT= 1; SLIMIT= 16

When you are running sessions/jobs under different accounts and wish to report on all those in a particular state, you again name the state as a :SHOWJOB command parameter:

State
↓

```

:SHOWJOB EXEC

JOBNUM  STATE  IPRI  JIN   JLIST      INTRODUCED  JOB NAME
#S745   EXEC           29   29         TUE  2:53P    DL,SPL.ALANG
#S746   EXEC           26   26         TUE  2:53P    CLI.AOPSYS
#S747   EXEC           42   42         TUE  2:53P    MAC.TECHPUBS
#S748   EXEC           47   47         TUE  2:53P    FIELD.SUPPORT

4 JOBS (DISPLAYED):
  4 SESSIONS
JOBFENCE= 2; JLIMIT= 1; SLIMIT= 16

```

Sometimes you may want to get a report on all sessions and jobs in the system. You may need this, for example, when you wish to send a message to a certain user (with the :TELL command) and want to determine if he is currently logged-on. Request your report by entering:

```

:SHOWJOB

JOBNUM  STATE  IPRI  JIN   JLIST      INTRODUCED  JOB NAME
#S745   EXEC           29   29         TUE  2:53P    DL,SPL.ALANG
#S746   EXEC           26   26         TUE  2:53P    CLI.AOPSYS
#S747   EXEC QUIET    42   42         TUE  2:53P    MAC.TECHPUBS
#S748   EXEC           47   47         TUE  2:53P    FIELD.SUPPORT
#S749   EXEC           34   34         TUE  2:55P    SPLT.ALANG
#S750   EXEC           28   28         TUE  2:55P    MIKE.DCA
#J396   INTRO    13    6S FASTLP   TUE  2:55P    SPLT.ALANG

7 JOBS:
  1 INTRO
  0 WAIT; INCL 0 DEFERRED
  6 EXEC; INCL 6 SESSIONS
  0 SUSP
JOBFENCE= 2; JLIMIT= 1; SLIMIT= 16

```

NOTE

In the above listing, the notation *QUIET* indicates a user who has disabled his terminal's facility for receiving messages from other users (:TELL command) and the Console Operator (=TELL command), as discussed in Section IX.

You can obtain other information relating to sessions and jobs currently logged-on, such as:

- The current availability and ownership of individual input/output devices connected to the system (requested via the :SHOWDEV command).
- The status and characteristics of input and output devicefiles currently existing (requested through the :SHOWIN and :SHOWOUT commands, respectively).

This information and the commands used to request it are described in Section VIII.

HOW MPE HANDLES RESOURCE ACCOUNTING

As each session or job progresses, MPE keeps track of the following system resources used by that session/job:

- Central processor time (in seconds).
- On-line terminal connect time for sessions (rounded upward to nearest minute).
- Permanent file space (in disc sectors).

MPE also monitors the collective use of these resources by all sessions and jobs under each account and group.

RESOURCE-USE LIMITS

When a System Manager creates an account — the basic “billable unit” to which resources are charged — he may establish limits for the maximum use of these resources under the account. For instance, he may specify that users running under this account cannot collectively accumulate more than 500 million seconds of central-processor time; if they attempt to do so, they are denied access to the account. Similarly, when an Account Manager creates a new group under his account, he can also place limits on resource usage. At both the account and group levels, the maximum limits that a manager can impose are approximately: 2 billion seconds for central-processor time, 2 billion minutes for connect time, and 2 billion sectors for file space. (This 2-billion figure is an approximation whose precise value is $(2^{31} - 1) = 2,147,483,647$.) If the System Manager does not specify any limit for a particular resource at the account level, no limit exists for that resource at this level. If the Account Manager does not specify any limit for such a resource at the group level, the limit (if any) defined at the account level also applies to the group. When he creates a group, the Account Manager cannot specify limits for the group that exceed those for the account; but if the System Manager later changes the limits for the account, the group then may possibly be left with limits that exceed the account limits.

When the System Manager creates an account, MPE automatically creates a public group named PUB under that account; all users of the account can read data and execute programs in files in this group. The limits for this group are initially the same as those for the account, but they may be changed later by the Account Manager.

If the System or Account Manager changes any limit for resource usage, this change becomes effective the next time that MPE is requested to check the limit.

RESOURCE ACCOUNTING

When you enter a session or job, MPE automatically checks to determine whether the limits on central-processor and connect time accumulated for your log-on account and group have already been exceeded by other sessions and jobs. If they have not, and if your session/job passes all other MPE access checks, it is granted access to the system. As the session/job progresses, MPE maintains running counts of the central-processor time, connect time (session only), and permanent file space it uses. If the session/job exceeds an account or group limit on central-processor or connect time, it is still allowed to continue; MPE only checks these limits at log-on time. But if the session/job makes a request to save a file and that action would exceed the account or group limit on permanent file space, MPE denies the request. (MPE always charges file space to the group that contains the file.)

NOTE

Your session or job continues to run even when it exceeds the limit on central-processor time specified for your log-on account and group because this limit is checked only at log-on time. But the session/job is aborted when it reaches any limit specified in the *TIME=cpusecs* parameter of the :HELLO or :JOB command, since this limit is checked dynamically during the course of the session/job.

When you log-off, MPE updates the resource usage counters at the account and group levels to reflect the amount of resources used by your session/job. If another user attempts to initiate a session or job and the central-processor or connect time limit has been reached at either the account or group level, MPE rejects that session/job.

Occasionally, you may wish to know how much central-processor time, connect time, and permanent file space has been accumulated under your log-on account and group, and what limits exist on the use of these resources. You typically want this information when you suspect that your session/job may encounter a limit — for instance when you are running a lengthy job on an account that has existed for several months or when you are building a large permanent file in a group already known to contain many such files. You could also use this information when using a different file group for the first time or when you frequently alternate between several different file groups, to familiarize yourself with the current status of these groups. You can display the total resource usage logged against your log-on account and group, and the limits on those resources, by entering the :REPORT command as shown below. The resulting display reflects file space used as of the present moment, but shows the central-processor and connect time as logged immediately before your session/job began. If you see that a limit is about to be reached, contact your System or Account manager about raising the existing limits or clearing the resource-use counters.

```
: REPORT
ACCOUNT          FILESPACE-SECTORS    CPU-SECONDS    CONNECT-MINUTES
  /GROUP          COUNT      LIMIT    COUNT      LIMIT    COUNT      LIMIT
TECHPURS         182        5000      164        **       38         **
  /XGROUP         0          5000         0          **         0          **
```

Double asterisk (**) denotes that no limits exist

NOTE

If you have the System Manager or Account Manager Capability, you can issue a more explicit form of this command, referencing particular accounts and groups. Additionally, with the System Manager Capability, you can re-set resource use counts for an account and all of its groups. These operations are discussed in *System Manager/System Supervisor Manual*.

MANAGING FILES

SECTION

VI

The general purpose of any computer system is to input, process, and output information. Under MPE, this information may be created and used by the operating system itself, by compilers or other subsystems, by user programs, or by users themselves. To handle all information in a uniform, efficient way, MPE treats it as groups of data called files. Specifically, *a file is a collection of information or data identified by a name and currently recognizable in the system by MPE.* Users unfamiliar with systems that handle information in this manner sometimes find it helpful to think of a file in the traditional way that people employed in business and commerce have for many years — as a storage cabinet containing information of various kinds. (See figure 6-1.) Instead of a filing cabinet, of course, MPE uses media such as disc, cards, and tape for storing the information. On any of these media, a file may contain MPE commands, system or user programs, or data — alone or in any combination. For instance, a card file containing a batch job might include commands to compile, prepare, and run a payroll-processing program written in COBOL, plus the program itself. The resulting object program, brought into memory from a file on disc, might read input from another disc file that contains wage and salary information for all employees on the company payroll; the program might also write new output to this same file during updating operations.

Within a file, all information is organized into units of related data called *logical records* that for most applications are similar in form, purpose, and content. The records in the file can be arranged in almost any order — alphabetically, numerically, chronologically, by subject matter, and so forth. For example, in the payroll file, each logical record could contain the wage and salary data related to a particular employee; there would be one record for each employee, and the records would be arranged in alphabetical order according to the employees' last names. Returning to the file cabinet analogy, the logical record would correspond to a sheet of paper serving as the employee's payroll record, stored in the cabinet. The logical record is the *smallest grouping* of data that MPE can directly address; you specify its length when you create the file. (Individual subsystems and user programs, however, can also recognize fields for data items within each record; for instance, a payroll record for an employee might contain a field for each of the following items: the employee's name, social security number, marital status, gross pay, tax exemptions, individual deductions, and net pay. Beyond this, programs can also recognize and manipulate individual words, eight-bit bytes, and bits within a byte.)

Data is transferred to and from a file in units called blocks; these are the basic units that are physically transmitted between main memory and the peripheral device on which the file resides. On disc and magnetic tape files, a block consists of one or more logical records; on files on other media, a block is normally equivalent to one logical record (unless you request input/output under the multi-record mode, discussed later in this section). In the file-cabinet analogy, a block is equivalent to a file folder that contains one or more payroll records, removed from the cabinet and returned to it as a set. These records have no particular relationship to one another, except that they are handled most easily when kept together in the folder.

To summarize the interrelation of files, logical records, and blocks:

- A *file* is a collection of records treated as a unit and recognized by a name.
- A *logical record* is a collection of fields treated as a unit, residing in a file.
- A *block* is a group of one or more logical records transmitted to or from a file by an input/output operation.

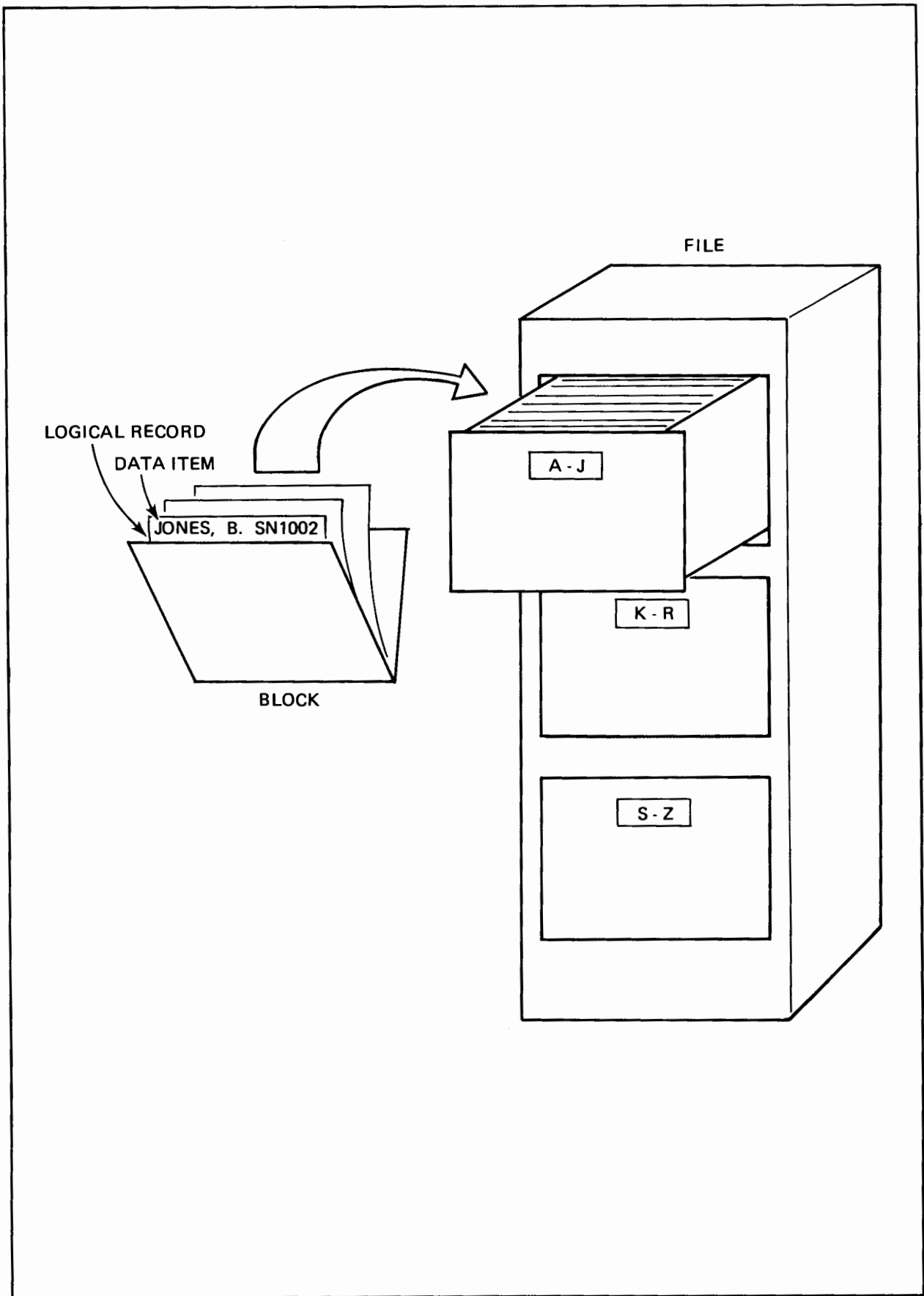


Figure 6-1. File Analogy

DISC FILES AND DEVICEFILES

MPE recognizes two basic types of files, classified on the basis of the media on which they reside when processed:

- *Disc files*, which are files residing on disc, immediately accessible by the system and potentially sharable by several sessions/jobs at the same time.
- *Devicefiles*, which are files currently being input to or output from any peripheral device *except* a disc. When information exists on such a device but is not being processed, MPE cannot recognize it as a file. Thus, information on cards is not identified as a file until the cards are loaded into the card reader and reading begins; data being written to a line printer is no longer regarded as a file when output to the printer terminates. A devicefile is accessed exclusively by the session or job that acquires it, and is *owned* by that session/job until the session/job explicitly releases it or terminates.

NOTE

Spooled *devicefiles*, although temporarily residing on disc, are considered devicefiles in the fullest sense because they are always originated on or destined for devices other than disc, and because you generally remain unaware of their storage on disc as an intermediate step in the spooling process. Whether they deal with spooled or unspooled devicefiles, your programs handle input/output as though the files reside on non-disc devices. The Console Operator, not the user, controls the spooling operation.

HOW MPE HANDLES FILE PROCESSING

Your programs access files by issuing intrinsic calls processed by the MPE File Management System. Generally, a program opens a file (through the FOPEN intrinsic), operates on it (through various other intrinsics that read, write, and update information or perform other functions), and finally closes it (through the FCLOSE intrinsic). In the file cabinet analogy, these steps are analagous to opening the cabinet; removing and reading or writing upon records and replacing them; and closing the cabinet. If you are programming in SPL, you must explicitly call the intrinsics that perform these operations from your program. But if you are writing a program in any other language such as COBOL, FORTRAN, or RPG, the compilers for those languages automatically issue the intrinsic calls for you. For instance, in a COBOL program, the OPEN statement implicitly issues the FOPEN intrinsic call to open a file; in a FORTRAN program, the first READ or WRITE statement that references a particular file issues FOPEN for that file. At your option, you can also call some intrinsics explicitly from languages such as FORTRAN and BASIC. The important fact to realize is that *some program – either your user program or the MPE subsystem that compiles it – must call the intrinsics that handle your files*. For details on calling intrinsics, see the appropriate language manuals and the *MPE Intrinsics Reference Manual*.

NOTE

If you are programming in a language other than SPL, you generally do not need to be aware of the detailed file operations performed and the intrinsic calls issued for you by the system. In most cases, the compilers use the descriptors you provide within your language statements to generate the appropriate files for you. If you are programming in SPL, however, or if you are writing special programs that demand the maximum power, efficiency, and flexibility of the File Management System, you will need some understanding of how this system operates. For this reason, an overview of general file-processing operations appears below. In this discussion, intrinsics are treated in a general way, adequate for those who need to know what functions they perform. For further information on specific calls, individual parameters, default values, and error codes, see *MPE Intrinsics Reference Manual*.

OPENING A FILE

When you open a file for access, the following operations occur, generally in the order shown:

1. Your program calls the FOPEN intrinsic to obtain access to the file. If your program is written in any language but SPL, the intrinsic call is supplied implicitly by the compiler that produces the object program. If this is an SPL program, however, you must explicitly supply the call in your source code. In the call, your program refers to the file by its *formal file designator* — the name by which the program formally recognizes the file. As an example, this name might be FILE1. The intrinsic call also specifies various characteristics that describe the file, or allows MPE to supply these characteristics by default. Some of these characteristics are:
 - The type of device on which the file resides.
 - The size of the logical records in the file.
 - The size of the blocks in which records are transmitted to and from the file.
 - The maximum number of records allowed in the file.
 - The type of access (read-only, write-only, and so forth) allowed those who use the file.
 - Whether the file is new to the system or has already been defined within it. (For old files, MPE already has the other information needed.)

In addition, the FOPEN intrinsic call also specifies other important information discussed throughout this section.

2. MPE reconciles the specifications in the FOPEN intrinsic with those that may be supplied in other sources, and merges this information. During this operation, the information in these sources takes the order of precedence shown below:
 - a. *The file label*, if this is a disc file that already exists in the system (an *old* disc file). This label appears in the first sector assigned to the file and contains such data as file designator, creation date, maximum number of logical records allowed, logical record size, block size, and more information to be described later. To read this label, MPE first searches the *session/job temporary file* directory for a pointer to the disc address of the label; this directory lists all files previously opened by your session or job that have not been explicitly deleted. If MPE does not locate the file in that directory, it then searches the *system file directory* which lists all files permanently saved in the system by any session or job. If the file cannot be found in either directory, MPE returns an error code to your program. The file information in the label overrides that from any other source.
 - b. *The parameter list of any MPE :FILE command that applies to this file*. You may use this command to re-specify some of the file's characteristics at the time you run your program, as discussed later in this section. Of course, this command always precedes the one that runs the program and references the file by the same formal designator used in the program. The information in this command overrides that supplied in the next two sources.
 - c. *The parameter list of your FOPEN call*.
 - d. *The default information supplied by MPE when no corresponding information is available in the previous three sources*. (Most MPE subsystems and user programs open files under most of these default values, allowing MPE to accept the primary burden of defining file characteristics.)

At this point, if the file is an old disc file, MPE also verifies your right to access the file under the file-access and security provisions specified in the file's label. If any of these verifications fail, MPE aborts the FOPEN intrinsic and returns an error code to your program.

The above hierarchy of precedence is illustrated graphically in figure 6-2.

3. MPE checks the type and characteristics of the device on which the file resides (if it is an *old* file) or on which it is to be created (if it is a *new* file — one that has not previously existed in the system). If this is a new disc file, MPE allocates disc space for it. The space is allocated as *extents*; each extent is an integral number of disc sectors made available to the file as it is needed. In the FOPEN request, you can specify the number of extents you wish allocated immediately, as well as the maximum number you wish to ultimately use.
4. MPE allocates space in main memory and on disc for various tables used to provide linkage between your program and the file. One of these tables is the *Available File Table* (AFT), residing in memory, that lists the files available to your session or job, and their locations and device types. At this point, MPE also allocates space for one or more buffers in main memory to be used as an interim storage area for records transmitted to and from the file. (You specify the size and number of buffers in your FOPEN request. The size of each buffer always equals the block size and is always sufficient to hold an integral number of logical records.)
5. If this is a new disc file, MPE creates a file label for it within your running program's stack (data area), and then writes this label to the first sector allocated for the file on disc. If this is an old disc file, MPE updates some of the information in the label, such as the date of current access.

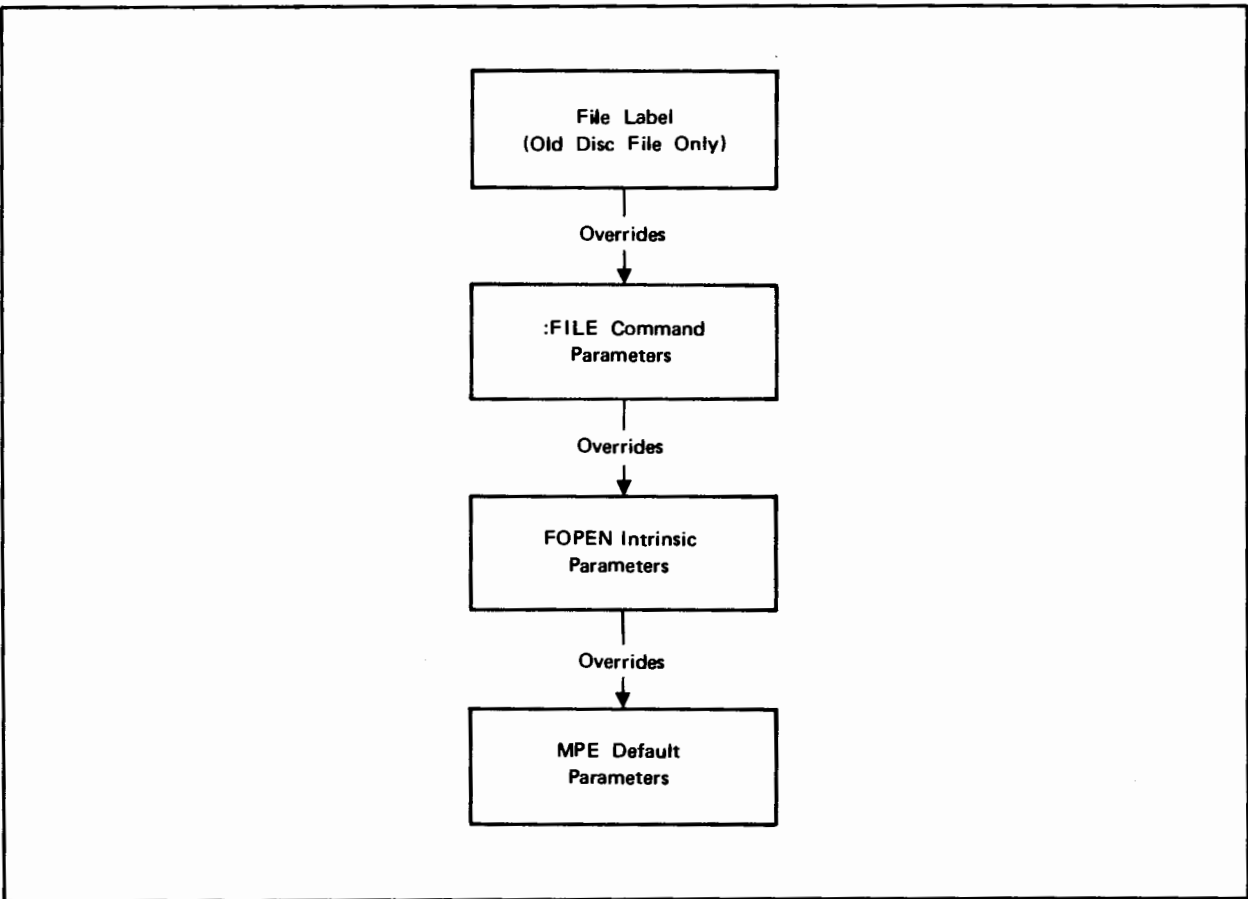


Figure 6-2. File Specification Hierarchy

6. Now, MPE opens the file, making it available to your program. If the file is a devicefile, MPE must allocate the corresponding device. The type of device on which the devicefile resides determines whether it is opened as a new or old (existing) file, and this factor is also reconciled with the information merged in Step 2 above. In general, devicefiles used only for input are opened as *old* files; those used only for output are opened as *new* files; and those used for both input and output are opened as combined *old/new* files. Thus, a file originating on a card reader would be an old file, one destined for a line printer would be a new file, and one associated with a terminal would be an old/new file.

NOTE

Unless you are performing unusual programmatic applications, you need not be aware of the operations described immediately below, since MPE handles these operations automatically. But for special applications, you may be interested in these functions.

- *For an old devicefile*, MPE searches the MPE *device directory* which lists all devices presently associated with files. A devicefile is listed in this directory when someone enters an MPE :DATA command, usually followed by file data, via the corresponding device. This command, which may name the file, makes it available to the system; if entered via a spooled device, the command spools the file to disc. The device, of course, must be configured to accept data in this fashion. Your FOPEN call can access the devicefile by specifying a formal file designator that

matches the one supplied in the :DATA command. (More information about this command appears on page 6-81.) If MPE does not find the requested devicefile in the device directory, it sends a message to the console operator requesting him to make the device available, either by loading data beginning with the :DATA command or by performing various console operations. Your program waits until this is done and then accesses the devicefile.

- *For a new devicefile*, MPE normally allocates the first device available, without requiring operator attention. But if this file requires special forms or other media (such as tape) for output, MPE transmits a message to the operator requesting him to mount the medium and waits until he completes this task. Then, MPE allocates the device.
- *For a combined old/new devicefile*, MPE treats the file according to the type of access specified in the FOPEN request. That is, if FOPEN specifies read access, the devicefile is opened as an *old* (input) devicefile; if FOPEN specifies write access, the devicefile is opened as a *new* (output) devicefile. If FOPEN specifies both read and write access, the devicefile is opened under two separate accesses (and file numbers) — one for an old file and another for a new one. If FOPEN specifies write (output) access but a :FILE command is entered altering this to read (input) access, the file is opened as an *old* file for read access. Devicefiles residing on a terminal or card reader/punch only permit your program to access them under the type of access assigned when they were opened. But devicefiles on magnetic tape can be accessed in any manner regardless of how they were opened; for example, even if a tape devicefile is opened as a new (input only) file, your program can both read it and write upon it.

In addition to classification as an old or new file, many other file characteristics that you specify in the FOPEN call for a devicefile are restricted by the physical nature of the corresponding device. As an example, you must open a card reader with read-only access and specify a buffer size equivalent to one logical record. If you enter any parameter that violates these restrictions, MPE overrides that parameter and supplies a proper one.

NOTE

To open a devicefile, you must possess the Non-Sharable Device File-Access Capability, assigned to most users by the Account Manager.

7. MPE now copies file-access information and pointers from your stack into the appropriate linkage tables and control areas, removes this information from the stack, and returns a unique *file number* to your program. This number can range from 1 to 255, and uniquely identifies the file to your program. This value is actually the index to the entry concerning the file in the Available File Table (AFT) for your session or job. In an SPL program, you obtain the file number through the normal conventions of the language, and use it in all subsequent intrinsic calls to reference the file. In programs written in other languages, the intrinsic calls implicitly generated by the language statements that access the file equate the file name to the file number automatically. This allows you to reference the file by its formal designator throughout the rest of the program, without concern for the file number.

NOTE

When MPE allocates disc space for a new file, it enters that file into the session/job temporary file directory. The files in this directory exist until the session/job terminates, at which time they are destroyed. Thus, to save a new file in the system, you must close it with permanent disposition (FCLOSE intrinsic) or declare it permanent via the MPE :SAVE command (page 6-63) before your session/job is over.

TRANSFERRING DATA TO AND FROM A FILE

Now your program is ready to write data to the file or read data from it. If the file is a disc file, this input/output can be conducted either sequentially or by direct access. If the file resides on any other type of device, however, input/output is always conducted sequentially. In sequential access, the records are input/output in consecutive order. In direct access, the records can be read or written in random order — for example, your program can read the tenth record in a file before it reads the first, or it can write to the eighth disc sector prior to writing to the first, and so forth.

DISC FILES. To write information to a file sequentially, your program calls the `FWRITE` intrinsic which transfers one logical record into the buffer established for the file. (If this is a new disc file that you have just opened, the only information in the file at this point is the file label.) Suppose that the `FOPEN` request specified that each logical record is 80 bytes (ASCII characters) long and that the buffer space is large enough to accommodate three logical records. When your program initially calls the `FWRITE` intrinsic, it moves the first logical record from your stack into the first location in the buffer. This record remains in the buffer until your program issues two more `FWRITE` calls to move the second and third records from the stack into the second and third positions in the buffer. Now, when the buffer is full, MPE automatically moves the records from the buffer to the file on disc. This transfer occurs automatically each time the buffer is full, with all logical records in the buffer moved to disc as a *block* of three logical records. This technique, illustrated in figure 6-3, is called *blocking*, and is considerably more efficient than transferring the individual logical records to disc in three separate input/output operations. Because the blocking operation, including input/output between buffer and file, is handled automatically by MPE, your program remains free of the actual record-handling details involved. On the disc, each block is written into a sector that is 256 bytes long. In this example, since the total block length is 240 bytes (3 logical records x 80 bytes), 16 bytes in each sector remain unused. On disc, in normal recording mode, blocks are either the same size as, or longer than, the logical records. (For some applications, however, you may wish to request the special *multi-record recording mode*, where logical records can span block boundaries. This mode, sometimes used for particular operations involving unbuffered data transfers, is discussed on page 6-36.)

To read information from the file sequentially, your program calls the `FREAD` intrinsic. With this operation, blocking is performed in the same way, except that the data is moved from the file to your stack.

To write or read information via direct access, your program calls the `FWRITEDIR` and `FREADIR` intrinsics respectively. Basically, the blocking mechanism operates in the same way as in sequential access, except that records can be read from or written to the file in random order.

For disc files, many other operations can be performed, such as requesting access and status information (`FGETINFO` intrinsic); requesting file error information (`FCHECK`); reading and writing user-defined labels (`FREADLABEL` and `FWRITELABEL`, respectively); updating information (`FUPDATE`); and spacing forward or backward on a file (`FSPACE`). These and other file intrinsics are discussed in *MPE Intrinsics Reference Manual*.

DEVICEFILES. In normal recording mode on magnetic tape devicefiles, blocks may contain one or more logical records; on any other devicefiles, blocks (and buffer sizes) always equal one logical record. For instance, in normal input/output mode, each block read from a card reader consists of one punched card; each block written to a line printer consists of one line of print, left-justified on the printed page. (In the special multi-record mode, however, input/output requests are not necessarily confined to block boundaries, as discussed on page 6-36.)

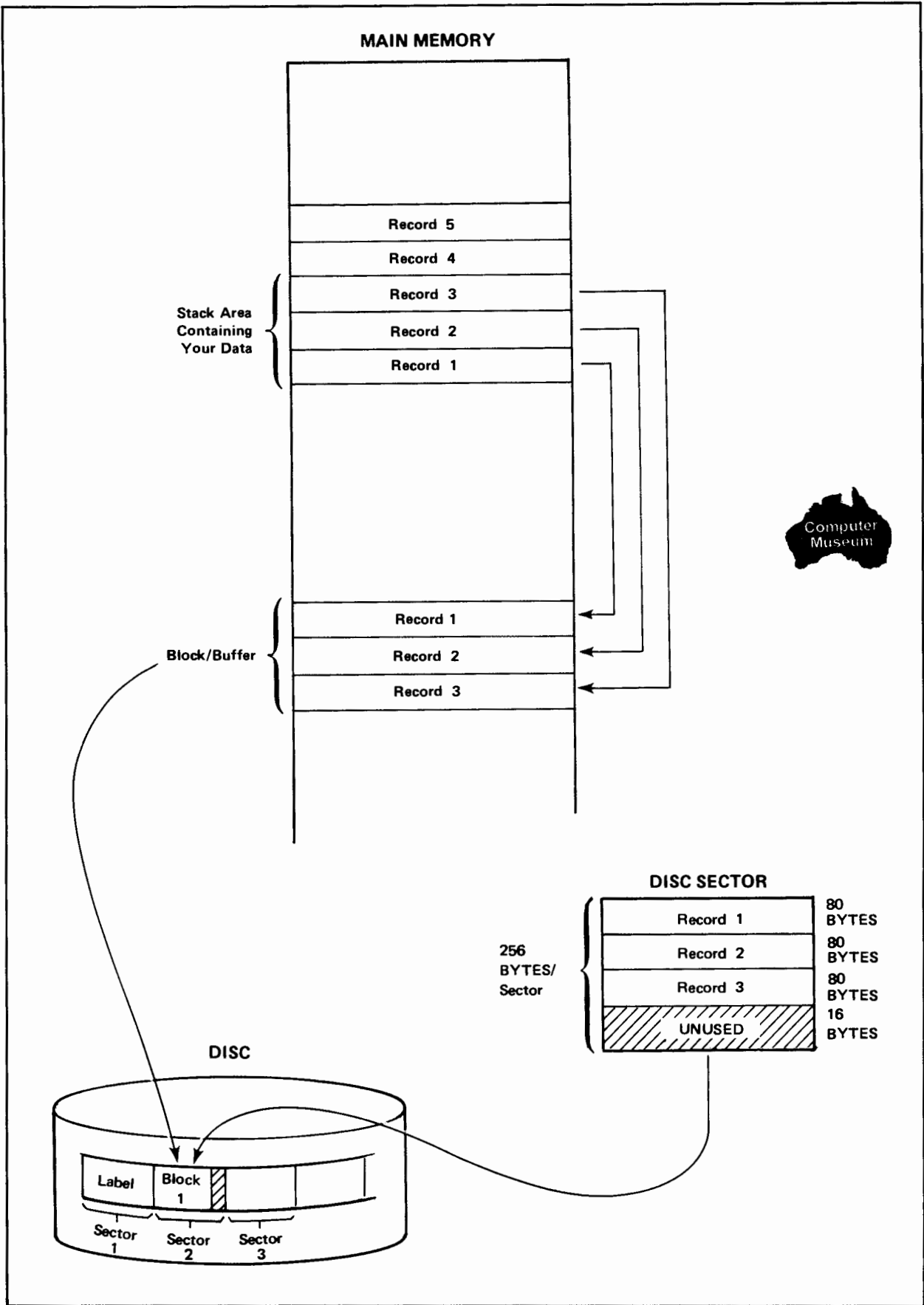


Figure 6-3. Sequential Writing to New Disc File

The interrelation of logical records and blocks in normal record mode on the various media is illustrated in figure 6-4. The flow of information between main memory and the file is essentially the same as described earlier for sequentially-accessed disc files.

If a file resides on magnetic tape, other programs within your session or job may access it but those run by other sessions/jobs may not do so until all programs within your session/job have closed the file. (Each FOPEN call for a file results in a separate, unique file number assigned to the file — in effect, a separate access of the file.) But if the file resides on any other type of non-disc device, only one running program at a time may access it even though other programs may be run by this session/job.

In the case of spooled devicefiles, your program assumes that all required devicefiles are constantly available to it and owned by it. Thus, input data is read from the originating device and stored before the program actually requires it, and output data is not written to the destination device until the program completes its file operations and closes the devicefile. There are, however, no other differences between spooled and unspooled devicefiles from the programming point of view.

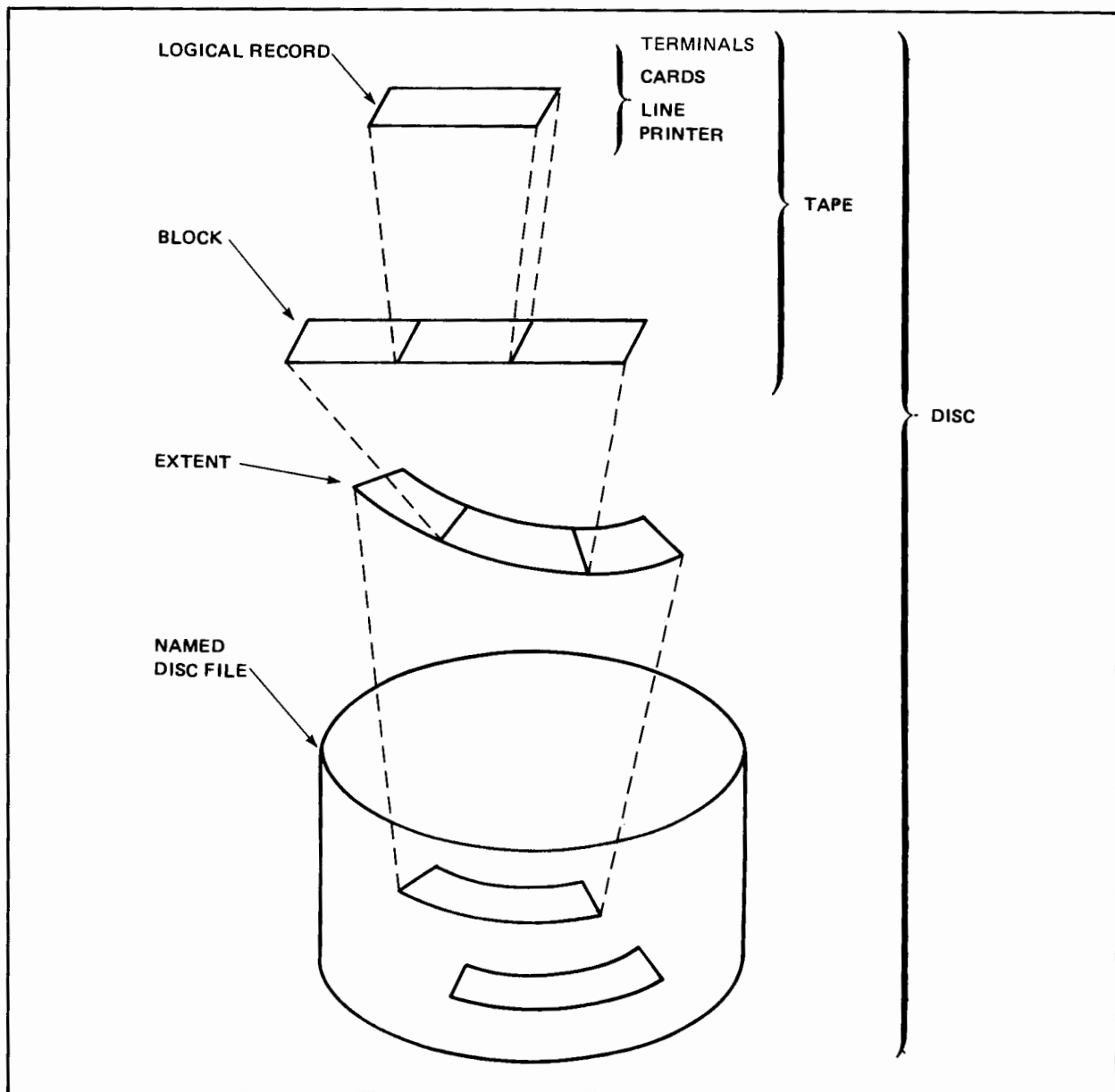


Figure 6-4. MPE File Structure

CLOSING A FILE

When your program has completed its input/output, it closes the file by calling the FCLOSE intrinsic. If this is a new disc file and the MPE default operations are allowed to take effect, FCLOSE destroys the buffers, pointers, and other system references that enable you to access the file. Thus, you must decide at this point whether you want to save the file in the system for future use or allow it to be released. This disposition is determined by an FCLOSE parameter. For programs written in languages other than SPL, the FCLOSE intrinsic call is implicitly inserted into the object program by the compiler, with the disposition determined by the compiler; you can ensure that such a file is saved by issuing the MPE :SAVE command (page 6-63). For SPL programs, since you explicitly specify FCLOSE, you can select the disposition desired by using the appropriate FCLOSE parameter or entering the :SAVE command. The following options are available:

- If you (or the compiler that creates the object program) declare the file *permanent*, MPE checks to determine if another file of the same name exists in the system. (Until this time, the name you assigned to the file did not need to be unique.) If a conflict exists, MPE returns an error indication to your program. If no conflict exists, MPE copies the file name and disc address from the file label into the *system file directory*. Files within this directory belong to the *system file domain*, and remain available until explicitly purged by their creators. Within this directory, files are listed according to the accounts to which they belong, and are organized into groups under those accounts. A new file is always assigned to the log-on account and group under which it was created. When entered in the system file directory, the file becomes an *old permanent file*, always available within the system.
- If you declare the file *session/job temporary*, MPE verifies the uniqueness of the file name among the files opened by your session/job and leaves the file name and disc address in the *session/job temporary file directory*. This directory lists all files opened at some time by the session/job that have not been deleted. Files within this directory remain available to your session or job until it terminates, at which time they are deleted. As long as the session or job is running, these files can be closed and reopened by any program within the session/job. With this disposition, a file becomes an *old temporary file*.
- If you declare the file *released*, its name and location are deleted from the session/job temporary file directory and the file is released from the system — it no longer exists as accessible data. The disc space used by the file is returned to the system.

An old disc file is closed with the same disposition it had when opened, unless you change this disposition with the FCLOSE call or :SAVE command.

A devicefile is closed with the same disposition it had when opened; this may be either session/job temporary (old files) or released (new files).

If FCLOSE specifies an incorrect or illegal disposition, the intrinsic fails and an error code is returned to your program. Thus, if you try to close a line printer as an old temporary file, your program receives an error code.

FILE DOMAINS

As noted earlier, the set of all permanent disc files in MPE is listed in the system file directory and is known as the *system file domain*. Within this domain, files are assigned to accounts and organized into groups under those accounts. The log-on process associates you with an account and group that provides the basis for your local file references. You may be required to supply *passwords* for the account and group to log on, but thereafter (if the default MPE file security provisions are in effect as is customary), you can:

- Have unlimited access to any file within your log-on or home group. If, however, the file is protected by a *lockword*, you must know this lockword. (You specify this lockword when you access the file, as directed on page 6-20).
- Read, and execute programs residing in, any file in the *public group* of your account, and in the *public group* of the system account.

Potentially, if the MPE file security provisions at the account, group, and file levels were all suspended, and you knew all account and group names and file lockwords, you could access any permanent file in the system once you logged on. Note that once you log on to an account and are associated with a group, you do not need to know the passwords for other accounts and groups to access files assigned to them — you only need to know their account and group names. But, if any of these files are protected by a file lockword, you must know this lockword.

NOTE

The details of the MPE file security system are discussed on pages 6-74 through 6-81.

For every session or job running in the system, MPE recognizes a unique *session/job temporary file domain*. This domain contains all temporary files opened and closed within the session or job that were not saved (declared permanent and entered in the system file directory). Files in your session or job temporary domain are listed in the session/job temporary file directory. They are deleted when the session or job terminates (if they are session/job temporary files), or when the creating program ends (if they are regular temporary files). Your session or job can access any files in this domain as long as they exist.

Spooled devicefiles are managed by the File Management System. They are written in a special format and may appear in the session/job temporary file directory but never in the system file directory.

SPECIFYING FILE CHARACTERISTICS

MPE's File Management System is designed to allow you great flexibility in the types of files your programs can access and the ways in which they can access them. For instance, it allows you to define new files on any type of device, block records for a disc or magnetic tape file for greatest efficiency, restrict a file to read-only, write-only, or other special access, share or restrict use of the file, and so forth. As a reflection of this flexibility, users often speak of files as being *device-independent*. By this, they mean that the name and characteristics assigned to a file when it is defined in a program do not restrict that file to residing on the same device each time the program is run. Thus, a file named SAMEFILE may be read from magnetic tape the first time you run the program, but read from disc or some other device on subsequent executions.

The fact that specifications for files used by any program are defined by the code making up that program initially appears to oppose the flexibility just described. These specifications are effectively "hard-coded" into the program. In an SPL program, they are defined by programmer in the FOPEN intrinsics that open the files; this would seem to allow the programmer sufficient control and flexibility when he writes a program, but would appear to lock him into an inflexible format each time he executes the program — unless he is willing to change his FOPEN statements and re-compile each time he wants to run the program with different file specifications. In programs written in other languages, where some or all file specifications are supplied by the compiler, it would seem that even this initial control and flexibility are denied the programmer. For instance, where a FORTRAN program defines a file named FTN05 as a file to be read from the card reader, there would seem to be no way to re-designate that file so that it could be read from disc.

To overcome this problem, MPE provides a link between the file specifications hard-coded into a program and the power and flexibility of the File Management System. This link is the MPE :FILE command, which enables you to re-specify the characteristics of files defined in your programs. To use the :FILE command for any file, you must open that file with a *formal file designator*. As noted earlier, the formal designator is the name by which your program recognizes the file. But it also performs another equally-important function: it provides a way for commands and code outside your program to reference the file — a kind of "handle to the outside world" for the file. The formal designator can thus be used to uniquely identify the file to the :FILE command.

NOTE

In SPL programs, you can open a new file without specifying a formal designator in the FOPEN intrinsic. When you do this, however, MPE provides a nameless file that can be used by your program but that *cannot* be referenced by the :FILE command; you *must* specify a *formal designator* in order to use the :FILE command.

When you are programming in a language other than SPL, and do not yourself write the FOPEN intrinsic calls for files used by your program, the :FILE command is the *only* way you can control or change the programmatic file specifications.

To illustrate how the :FILE command operates, suppose that you are about to run a COBOL program named MYPROG that, in its Data Division, defines an input file on cards named CARDFILE. In this file, each logical record contains 80 characters and is equivalent to one block. The hard-coded file specification in the program appears as follows:

```

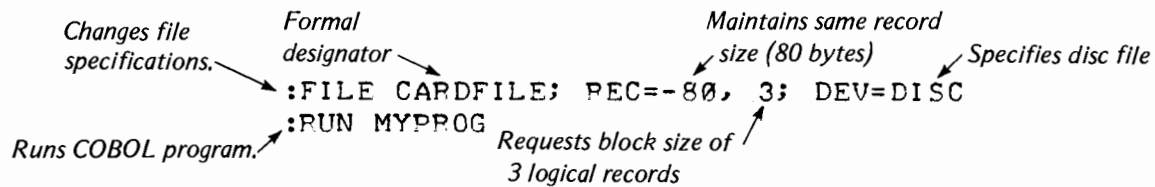
      .
      .
      .
DATA DIVISION.
FILE SECTION.
FD CARDFILE
   BLOCK CONTAINS 1 RECORDS
   DATA RECPD IS MYDATA
   RECORDING MODE IS F
   LABEL IS OMITTED
   RECORD CONTAINS 80 CHARACTERS.
      .
      .
      .

```

Annotations:

- Block size (1 record per block), intended for card input (points to 1 RECORDS)
- File name (points to CARDFILE)
- Record type (fixed length) (points to F)
- Logical record size (80 bytes) (points to 80 CHARACTERS)

Although this program was designed to accept its input from punched cards, there may be occasions where you wish to read this input from a disc file. In such cases, you may also wish to read these records in blocks of three logical records each. Rather than re-code and re-compile the program, you could use the :FILE command as follows to change the file specifications:



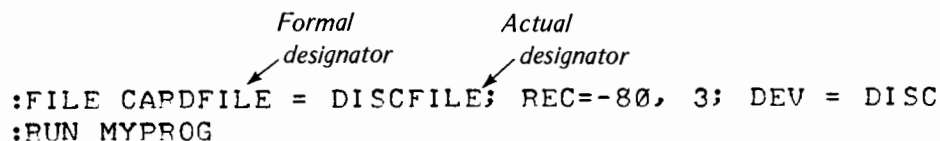
This illustrates the primary function of the :FILE command:

To allow you to change the specifications for files, including the devices on which they are input or output, at the time you run your program, overriding specifications supplied in the FOPEN intrinsic.

When the COBOL program executes an OPEN statement for the file named CARDFILE, this statement implicitly calls the FOPEN intrinsic, normally using the file specifications supplied in the File Description Section of the Data Division. But, before the file is actually opened, MPE checks to determine if your session or job has issued a :FILE command referencing the formal file designator CARDFILE. Because such a command exists, MPE uses the specifications supplied in that command to override those supplied by your program. (The :FILE command parameters included in this example are explained in greater detail later in this section.)

Although the formal file designator is the name by which your *program* recognizes the file, there must also be some means by which the *system* (MPE) also can recognize the file, allowing it to be referenced by various commands and programs. For an old disc file, this is the file name contained in the file label and in the system or session/job temporary file directory. For a devicefile, it is the name optionally supplied in the :DATA command and copied into the device directory. For a new disc file, it is the name you supply when you open the file; this name is then copied into the appropriate directory and entered in the file label. Whether it applies to a disc or devicefile, this is the *real* name that identifies the file to the *system*. It is called the *actual file designator*.

In the previous example, you assumed that the disc file used contained in its label the actual designator CARDFILE. Thus, the actual designator was the same as the formal designator used in the program. But, if the label had identified the file by some other name, — say DISCFILE — you would have needed to equate the formal designator to the actual designator in the :FILE command, as follows:



In this way, the :FILE command, by equating the formal and actual designators, provides a linkage between the File System label or name for a file and your program's name for that file.

Looked at from a slightly different point of view, this example also illustrates another purpose of the :FILE command:

To allow you to write programs that reference files whose names (and characteristics) you may not yet know at the time you are coding.

As an example, the original author of the program MYPROG, noted above, may not have realized that his program would sometime accept its input from a file named DISCFILE, stored on disc. This feature allows you to remain uncommitted to specific disc files or devices until run-time, when you equate their names with programmatic references.

An illustration of the linkage provided by equating formal and actual file designators via the :FILE command appears in figure 6-5. This illustration shows an SPL program that includes a direct call to FOPEN referencing the formal designator MYFILE, and a :FILE command that equates this name to the actual designator YOURFILE.

As a further example, suppose you are coding a program that will accept input from a disc file whose actual designator you may not know at this time. In your program, you can refer to the file by an arbitrary formal designator — say, ANYFILE. When you run your program, you discover that the creator of the input file has copied it to disc as a permanent file identified by the actual designator REALFILE. To access this file, you issue the following :FILE command:

```

      Formal
      designator
      ↙
:FILE ANYFILE = REALFILE ← Actual
:RUN MYPROGX                designator
```

Before your program opens the file, MPE searches for a :FILE command that equates the formal designator to an actual designator and executes that command, re-directing your program to REALFILE. (Because you wished to use all specifications in the file label for REALFILE, you did not need to declare any other parameters in the :FILE command.)

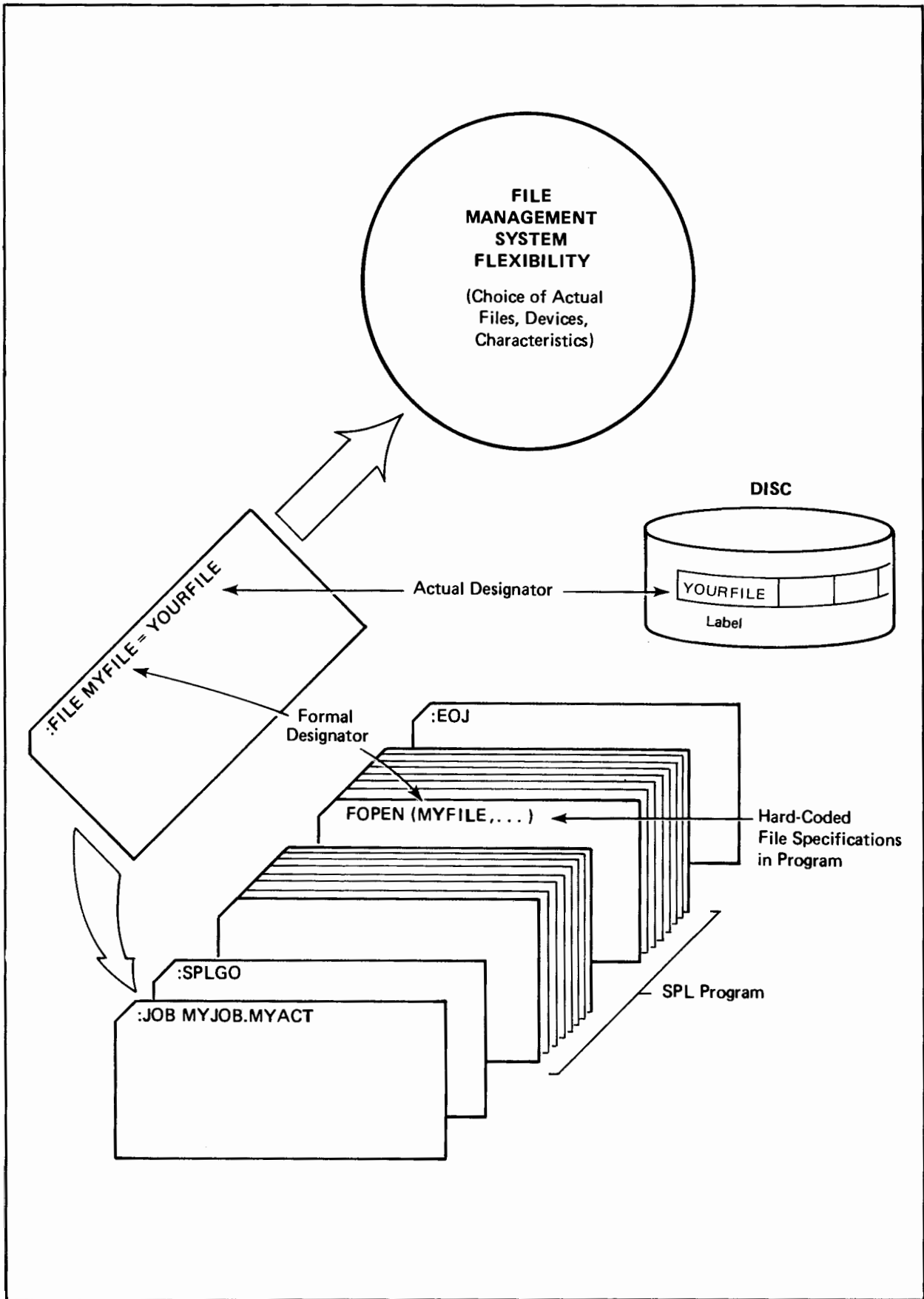


Figure 6-5. Formal vs Actual File Designators

SUMMARY OF GENERAL RULES

Earlier in this section, you saw that MPE obtains file specifications from a hierarchy of sources. It is important to bear this hierarchy in mind whenever you use the File Management System. To review the hierarchy:

1. *The file label is the overriding source if the file is an old disc file.* Thus, if you issue a `:FILE` command equating the formal designator `FILEX` to the actual designator `FILEY`, and request a blocking factor of 3 records per block when the label specifies 2 records per block, your program will indeed access `FILEY` — but the records will be read at 2 records per block. No error code is returned to your program.
2. *The `:FILE` command overrides the `FOPEN` intrinsic and the MPE defaults for new disc files and devicefiles.* If you omit a parameter from the `:FILE` command, the corresponding `FOPEN` parameter (or its default) takes effect.
3. *The `FOPEN` intrinsic overrules the MPE defaults for new disc files and devicefiles.*
4. *The MPE defaults take effect only when no other specification source (label, `:FILE` command, or `FOPEN` parameter) exists.*

Whether a file is accessed by a user program, an MPE subsystem such as a compiler, or an MPE procedure that executes a command (command executor), the accessing program always refers to that file by its formal file designator. The author of the program determines what the formal designator is when he (explicitly or implicitly) invokes `FOPEN`. When the program is executed, this formal designator is always equated to an actual file designator. The formal designator, hardcoded into the program, never changes from run to run. But the actual designator can be altered whenever you desire, simply by changing the corresponding parameter in the `:FILE` command. For instance, assume you have a program named `MYPROGZ` that you wish to execute three times; on each run, `MYPROGZ` is to read input from three different disc files (with the actual designators `FILE1`, `FILE2`, and `FILE3`). If the formal file designator used for the input file by the program is `FILEN`, you could precede each run by the following `:FILE` commands. (In all three disc files, the record formats used must be the same.)

```
      .  
      .  
      .  
      :FILE FILEN=FILE1  
      :RUN MYPROGZ  
      :FILE FILEN=FILE2  
      :RUN MYPROGZ  
      :FILE FILEN=FILE3  
      :RUN MYPROGZ  
      .  
      .  
      .
```

The specifications in the `:FILE` command do not take effect until your program is running and opens the file referenced. The `:FILE` command specifications then hold throughout the entire session/job unless superseded by another `:FILE` command or revoked by a `:RESET` command. When your session or job terminates, however, all `:FILE` command specifications are cancelled.

When two or more `:FILE` commands referencing the same formal designator appear in a session/job, the last command encountered takes effect.

If your program names a formal file designator which you have not equated to an actual designator via a `:FILE` command, the formal designator itself is used as the actual designator, and all file characteristics specified (explicitly or by default) within your program apply. For instance, in the program MYPROGZ described above, the formal file designator FILEN would also be used as the actual file designator, specifying an old input file. To open the file, MPE would search in vain for an old disc file named FILEN and an error code would be returned to your program. (If your program had described FILEN as a new disc file, MPE would open a new file named FILEN on disc with the same actual and formal file designators.)

Occasionally, you may write a program that references files whose specifications must always remain the same. For example, many application programs require files that should allow read-access but prohibit write-access. If you are writing such a program in SPL, or are calling FOPEN directly from some other language, you can specify within your FOPEN call that MPE should ignore any `:FILE` equation that references the file you are opening. This effectively locks-out control of the file characteristics by other users, allowing them to use your file *only* as your program permits. For further information on disallowing `:FILE` equations, see *MPE Intrinsic Reference Manual*.

SPECIFYING FILE DESIGNATORS

MPE recognizes two general classes of files:

- *User-Defined Files*, which you or other users define, create, and make available for your own purposes, and
- *System-Defined Files*, which MPE defines and makes available to all users to indicate standard input/output devices.

These files are distinguished by the file names and other descriptors (such as group or account names) that reference them, as discussed below. You may use both the file name and descriptors, in combination, as either formal designators within your programs or as actual designators that identify the file to the system. Generally, however, most programmers use only arbitrary names as formal designators, and then equate them to appropriate actual file designators at run-time. In such cases, the formal designators (user file names) contain from 1 to 8 alphanumeric characters, beginning with a letter; the actual designators include a user or system file name, optionally followed by a group name, account name, and/or security lock-word, all separated by appropriate delimiters. This technique facilitates maximum flexibility with respect to file references, and permits easy, terse coding. It is used in most of the examples that appear in this manual.

USER-DEFINED FILES. You can reference any user-defined file by writing its name and descriptors in the filereference format. Syntactically, this format is:

filename [*/lockword*] [*.groupname*] [*.accountname*]

When you reference a file that belongs to your log-on account and group, you use the *filereference* format in its simplest form, which includes only a file name that may range from 1 to 8 alphanumeric characters, beginning with a letter. In the following examples, both formal and actual designators appear in this format:

```

      Formal
      designator
      :FILE ALPHA = BETA ← Actual
      :FILE REPORT = OUTPUT ← designator
      :FILE X = AL126797
      :FILE PAYROLL = SECFL

```

NOTE

Because a file reference is always qualified, in the appropriate directory, by the names of the group and account to which the file belongs, you need ensure only that the file's name is unique within its group. For instance, if you create a file named FILX under GROUPA and ACCOUNT1, the system will recognize your file as FILX.GROUPA.ACCOUNT1; a file with the same file name, created under a different group, could be recognized as FILX.GROUPB.ACCOUNT1.

File groups serve as the bases for your local file references. Thus, when you log-on, if the normal MPE file security provisions are in effect, you have unlimited access to all files assigned to your log-on group and your home group. Furthermore, you are permitted to read, and execute programs residing in, the Public Group of your log-on account. This group, always named PUB, is created under every account to serve as a common file base for all users of the account. Additionally, you may also read and execute programs residing in the Public Group of the System Account. This is a special account available to all users on every system, always named SYS.

When you reference a file that belongs to your log-on account but not to your log-on group, you must specify the name of the file's group within your reference. In this form of the *filereference* format, the group name appears after the file name, separated from it by a period. Embedded blanks within the file or group names, or surrounding the period, are prohibited. As an example, suppose your program references a file under the name LEDGER, which is recorded in the system by the actual designator GENACCT. This file belongs to your home group, but you are logged on under another group when you run the program. To access the file, you must specify the group name as follows:

```
:FILE LEDGER = GENACCT.XGROUP ← Group name
:PUN MYPROG ← Program file (in log-on group)
```

As another example, suppose you are logged on under the group named XGROUP but wish to reference a file named X3 that is assigned to the Public Group of your account. If your program refers to this file by the name FILLER, you would enter:

```
:FILE FILLER = X3.PUB
```

When you reference a file that does not belong to your log-on account, you must use an even more extensive form of the *filereference* format. With this form, you include both group name and account name. The account name follows the group name, and is separated from it by a period. Embedded blanks are not permitted. As an example, suppose you are logged-on under the account named MYACCT but wish to reference the file named GENINFO in the Public Group of the System Account. Your program references this file under the formal designator GENFILE. You would enter:

```
:FILE GENFILE = GENINFO.PUB.SYS
```

NOTE

You can only create a new file within your log-on account. Therefore, if you wish to have a new file under a different account, you log-on to the other account and create the file in that account and group.

Lockwords. When you create a disc file, you can assign to it a lockword that must thereafter be supplied (as part of the *filereference* format) to access the file in any way. This lockword is independent of, and serves in addition to, the other MPE security provisions governing the file.

You assign a lockword to a new file by specifying it in the *filereference* parameter of the :BUILD command or the *formaldesignator* parameter of the FOPEN intrinsic used to create the file. For example, to assign the lockword SESAME to a new file named FILEA, you could enter the following :BUILD command. (A complete discussion of the :BUILD command, used to create new disc files, appears later in this section.)

```
:BUILD FILEA/SESAME ← Lockword
```

From this point on, whenever you or another user reference the file in an MPE command or FOPEN intrinsic, you must always supply the lockword. It is important to remember that you need the lockword even if you are the creator of the file. Lockwords, however, are required only for old files on disc.

When referencing a file protected by a lockword, supply the lockword in the following manner:

- In batch mode, as part of the file designator (*filereference* format) specified in the :FILE command or FOPEN intrinsic call used to establish access to the file. Enter the lockword after the file name, separated from it by a slash mark. Neither the file name nor the lockword should contain embedded blanks. Additionally, the slash mark (/) that separates these names should not be preceded nor followed by blanks. The lockword may contain from 1 to 8 alphanumeric characters, beginning with a letter. If a file is protected by a lockword and you fail to supply that lockword in your reference, you are denied access to the file. In the following example, the old disc file XREF, protected by the lockword OKAY, is referenced:

```
:FILE INPUT = XREF/OKAY ← Lockword
```

- In session mode, also as part of the file designator specified in the :FILE command or FOPEN intrinsic call that establishes access to the file, using the same syntax rules described above. If a file is protected by a lockword but you supply no lockword when you open the file, MPE interactively requests you to supply the lockword as shown in the example below:

```
LOCKWORD: YOURFILE.YOURGROUP.YOURACCT?
```

NOTE

On terminals with the ESC (Escape) key facility, you can inhibit the echo facility (and thus suppress printing of the lockword) by pressing the ESC and ; keys before you enter the lockword in response to the above prompt. After you enter the lockword, you can restore echoing by pressing the ESC and : keys. Where the terminal does not permit dynamic half-duplex operation, MPE prints an eight-character mask following this prompt and returns the carriage to the beginning of the mask, allowing you to conceal the lockword by entering it on top of the mask.

Always bear in mind that the file *lockword* relates only to the ability to access files, and not to the account and group *passwords* used to log-on. Three examples of :FILE commands referencing lockwords are shown below; the last command illustrates the complete, fully-qualified form of the *filereference* format.

```

:FILE AFILE = PAYROLL/X229AD ← Lockword
:FILE BFILE = GOFILE/Z22.GP07 ← Lockword
:FILE CFILE = FILEM/LOCKZ.GROUPN.ACCTO ← Lockword

```

NOTE

In no case must any file designator written in the *filereference* format exceed 35 characters, including delimiters.

At any time, a file may have only one lockword. You can change the lockword by using the :RENAME command (page 6-60) or the FRENAME intrinsic (*MPE Intrinsic Reference Manual*). You can also initially assign a lockword to an existing file with this command or intrinsic. To do either of these tasks, however, you must be the creator of the file.

Passing Files. Programmers, particularly those writing compilers or other subsystems, sometimes create a temporary disc file that can be automatically passed to succeeding MPE commands within a session or job. This file is always created under the special name \$NEWPASS. When your program closes the file, however, MPE automatically changes its name to \$OLDPASS (and deletes any other file named \$OLDPASS in the session/job temporary file domain). From this point on, your commands and programs reference the file as \$OLDPASS. Only one file named \$NEWPASS and/or one file named \$OLDPASS can exist in the session/job domain at any one time. To illustrate how file passing works, consider an example where two programs, PROG1 and PROG2, are executed. PROG1 receives input from the actual disc file DSFIL (through the programmatic name SOURCE1) and writes output to an actual file \$NEWPASS, to be passed to PROG2. (\$NEWPASS is referenced programmatically in PROG1 by the name INTERFIL.) When PROG2 is run, it receives \$NEWPASS (now known by the actual designator \$OLDPASS), referencing that file programmatically as SOURCE2. Note that only *one* file can be designated for passing.

```

.
.
.
:FILE SOURCE1=DSFIL
:FILE INTERFIL=$NEWPASS ←
:RUN PROG1
:FILE SOURCE2=$OLDPASS ← Same file
:RUN PROG2
.
.
.

```

More examples of file-passing appear in Section VII, under the discussion of compiling, preparing, and executing programs.

Back-Referencing Files. Once you establish a set of specifications in a :FILE command, you can apply those specifications to other file references in your session or job simply by using the file's formal designator, preceded by an asterisk (*), in those references. For instance, suppose you use a :FILE command to establish the specifications shown below for the file FILEA, used by program PROGA. You then run PROGA. Now, you wish to apply those same specifications to the file FILEB, used by PROGB, and run that program. Rather than re-specify all these parameters in a second :FILE command, you can simply use :FILE to equate the FILEA specifications to cover FILEB, as follows:

```
:FILE FILEA; DEV=MT; REC=-80, 4, V; BUF=4  Establishes specifications.
:RUN PROGA                                Runs Program A.
:FILE FILEB = *FILEA                       Back-references specifications for FILEA.
:RUN PROGB                                 Runs Program B.
```

This technique is called back-referencing files, and the files to which it applies are sometimes known as *user- pre-defined files*. Whenever you reference a pre-defined file in an MPE command, you *must* enter the asterisk before the formal designator if you want the pre-definition to apply. As an example, you would do this when using the Editor in session mode when you wish to transmit the Editor's off-line listings to a device other than a line printer (such as a magnetic tape unit). You specify the destination file name as a parameter in the :EDITOR command that invokes the Editor, but you define the file in a previous :FILE command. You then back-reference the definition by using the asterisk in the :EDITOR command, as shown below:

```
:FILE L; DEV=TAPE  Specifies file named L as a tape unit.
:EDITOR *L        Runs Editor, directing off-line listings to file L, now established as a line printer.
```

SYSTEM-DEFINED FILES. System-defined file designators indicate those files that MPE uniquely identifies as standard input/output devices for sessions and jobs. These designators are described in table 6-1, below. When you reference them, you use only the file name; group or account names and lockwords do not apply.

Table 6-1. System-Defined File Designators

FILE DESIGNATOR/NAME	DEVICE/FILE REFERENCED
\$STDIN	The standard session or job input device from which your session/job is initiated. For a session, this is always a terminal. For a job, it is most often a card reader. Input data images in this file should not contain a colon in column 1, since this indicates the end-of-data. (When data is to be delimited, use the :EOD command, which performs no other function.)
\$STDINX	Same as \$STDIN, except that MPE command images (those with a colon in column 1) encountered in a data file, are read without indicating the end-of-data. (However, the commands :EOD and :EOF: (and in batch jobs, the commands :JOB, :EOJ, and :DATA) are exceptions that always indicate end-of-data but are otherwise ignored in this context; they are <i>never</i> read as data.) This file name is often used by interactive subsystems and programs to reference the terminal as an input file.
\$STDLIST	The standard session or job listing device, nearly always the terminal for a session and customarily a printer for a batch job.
\$NULL	The name of a non-existent "ghost file" that is always treated as an empty file. When referenced as an input file by a program, that program receives an end-of-data indication upon each access. When referenced as an output file, the associated write request is accepted by MPE but no physical output is actually done. Thus, \$NULL can be used to discard unneeded output from a running program.

As an example of how to use some of these designators, suppose you are running a program that accepts input from a file programmatically defined as INPUT and directs output to a file programmatically defined as OUTPUT. Your program specifies that these are disc files, but you wish to re-specify these files so that INPUT is read from the standard input device and OUTPUT is sent to the standard listing device. You could enter the following commands:

```
:FILE INPUT = $STDIN
:FILE OUTPUT = $STDLIST
:RUN MYPROG
```

INPUT/OUTPUT SETS. All file designators can be classified as those used for input files (*Input Set*) and those used for output files (*Output Set*). For your convenience, these sets are summarized in tables 6-2 and 6-3.

Table 6-2. Input Set

FILE DESIGNATOR	FUNCTION/MEANING
\$STDIN	Session/job input device.
\$STDINX	Session/job input device with commands allowed.
\$OLDPASS	Last \$NEWPASS file closed.
\$NULL	Constantly-empty file that returns end-of-file indication when read.
<i>*formaldesignator</i>	Back-reference to previously-defined file.
<i>filereference</i>	File name, and perhaps account and group names and lock word. Indicates an <i>old</i> file. May be a session/job temporary file created in this or a previous program in current session/job, or a permanent file saved by any program or :BUILD or :SAVE command in any session/job.

Table 6-3. Output Set

FILE DESIGNATOR	FUNCTION/MEANING
\$STDLIST	Session/job listing device.
\$OLDPASS	Last file passed.
\$NEWPASS	New temporary file to be passed.
\$NULL	Constantly-empty file that returns a successful indication whenever information is written to it.
<i>*formaldesignator</i>	Back-reference to previously-defined file.
<i>filereference</i>	File name, and perhaps account and group names and lock word. Unless you specify otherwise, this is a temporary new file residing on disc that is destroyed on termination of the creating program. If closed as a session/job temporary file, it is purged at the end of the session/job. If closed as a permanent file, it is saved until you purge it.

SEARCHING FILE DOMAINS

When your program opens a file, it can specify in the FOPEN call that the file is one of the following:

- A new file to be created at this point; MPE need not search its directories for this file.
- An old session/job temporary file; MPE must search the session/job temporary file domain (directory) for this file.
- An old permanent file; MPE must search the system file domain (directory) for this file.
- An old file to be located by first searching the session/job temporary file domain and then, if this fails, searching the system file domain.

If your program does not specify the domain to be searched, MPE treats the file as a new file by default.

If you wish to override the programmatic specification or default value, you can do so by entering a :FILE command that specifies one of the following parameters: *NEW* to indicate a new file, *OLDTEMP* to indicate an old file in the session/job temporary file domain, and *OLD* to indicate an old permanent file in the system file domain. You might wish to do this, for instance, if you had an RPG program (named REPGEN) that opened a new disc file named REPPFILE the first time it was run, wrote output to this file, and saved it as a permanent file. On subsequent executions, you would want to open this file as an old file, directing MPE to search for it in the system file domain so that your program could write additional output to the file. You could override the programmatic specification for a new file by entering:

```

      :
      :
      :
      :FILE REPPFILE, OLD
      :RUN REPGEN
      :
      :
  
```

In some cases, the domain you can specify for a file may be restricted by the type of device on which the file resides. The domains permitted are summarized in table 6-4.

Table 6-4. File Domains Permitted

DEVICE TYPE	DOMAIN
Disc	NEW, OLD, or OLDTEMP
Card Reader	OLD only
Paper Tape Reader	OLD only
Terminal	NEW or OLD
Printing Reader Punch	NEW or OLD
Synchronous Single-Line Controller	NEW or OLD
Programmable Controller	NEW or OLD
Magnetic Tape Drive	NEW or OLD
Line Printer	NEW only
Paper Tape Punch	NEW only
Plotter	NEW only

DEVICES

Devices required by files are allocated automatically by MPE. You can specify these devices by class (such as any card reader or line printer) or by a logical device number related to a particular device (such as a specific line printer). (A unique logical device number is assigned to each device when the system is configured.) Regardless of what device a particular file resides on, when your program requests to read that file, it references the file by its formal designator. MPE then determines the device on which the file resides, and its disc address if applicable, and accesses it for you. When your program writes information to a file destined for an output device such as a line printer, again the program refers to the file by its formal designator. MPE then automatically allocates the required device to that file. Throughout its life, every file remains *device-independent* — that is, it is always referenced by the same formal file designator regardless of where it currently resides.

Both the device class name and the logical device number associated with a device are determined by the System Supervisor or Console Operator when he adds the device to the system. The device class name is an arbitrary name that can be allocated to more than one device. The logical device number, however, is unique for each device; it may range from 1 to 255. As an example, devices might be configured with the class names and logical device numbers shown in Table 6-5.

Table 6-5. Device Configurations

DEVICE	LOGICAL DEVICE NO.	DEVICE CLASS NAME
System Disc (Required)	1	SYSDISC
Disc File	2	DISC
Card Reader	5	CARD
Line Printer	6	LP, PRINTER
Magnetic Tape	7	TAPE, TAPE0
Magnetic Tape	8	TAPE, TAPE1
Magnetic Tape	9	TAPE, TAPE2
Magnetic Tape (Job-Accepting)	10	JOBTAPE
Line Printer	11	LP
Console	20	CONSOLE
Terminal	21	TERM
Terminal	22	TERM

In this configuration, the card reader is assigned logical device number 5 and device class name CARD. In this case, you could make a unique reference to this device by using either the logical device number 5 or the class name CARD (since no other device shares this class name) when you open the file. In the case of a magnetic tape unit, you could make specific references to logical devices 7, 8, or 9, or to the device classes TAPE0, TAPE1, or TAPE2, respectively. But if you are willing to use any magnetic tape unit, you could make a non-specific reference to the class name TAPE, which would provide the first tape unit available for your file.

When an SPL program opens a file, it can specify any device for that file in the FOPEN intrinsic; if it specifies no device, the class name DISC is assigned by default. Programs written in other languages often restrict the devices you can use for certain files. For instance, a FORTRAN program always equates the file named FTN05 to the standard input device, and that named FTN06 to the standard listing device. In many cases, if you do not or cannot specify a device for a file in such programs, the program assumes the system default — the class name DISC.

You can, however, override the programmatic device specifications by using the :FILE command to specify different devices. For example, suppose you plan to use the BASIC interpreter from a terminal and wish to direct your program listing to any line printer rather than the subsystem default device (which is the standard listing device, your terminal). You first define the listing file, arbitrarily named PRINTER, as a line printer (class name LP) in a :FILE command. After you issue the :BASIC command to invoke the interpreter, you enter your BASIC program, which includes a LIST command that directs output to the file named PRINTER, which is now recognized as a line printer.

```
.
.
.
:FILE PRINTER; DEV=LP  Defines PRINTER as Line Printer file.
:      Device class name
:BASIC  Invokes BASIC Interpreter.
.
.
.
>10 FOR I = 1 TO 10
.
.
.
>LIST, OUT=PRINTER  Transmits output to PRINTER.
.
.
.
```

(The :BASIC command is discussed in detail on page 7-14.)

If a file is a spooled devicefile, you can assign an output priority to the file analogous to the *outputpriority* specified for job listing output in the :JOB command (page 4-9). The priority can range from 1 (lowest) to 13 (highest). The *outfence* established by the Console Operator relates to the priority for this file in the same way as that for the job listing file. That is, spooled output files with priorities lower than or equal to the outfence are not printed or punched until the outfence is lowered or the priorities are raised by the Console Operator. Suppose you are running a program that will print an extensive output file at a time when the computer is left unattended. To safeguard against problems arising from the printer jamming or running out of paper while it prints the file, you could specify an output priority less than the current outfence (8), and request the Operator to lower the outfence when he returns to the machine room. When this is done, your file can be transmitted from disc to printer. You might specify the priority as follows:

```
:FILE LONGFILE; DEV=LP, 6  ← Output priority
:RUN PROGX
```

If a file is directed to a spooled output device and you should require more than one copy of the file, you can request this by using the :FILE command. This feature is particularly useful for COBOL or RPG programs that generate reports that must be produced in multiple copies. To illustrate this feature, the following :FILE command requests four copies of a report produced on the line-printer file LISTREP, by the program REPGEN:

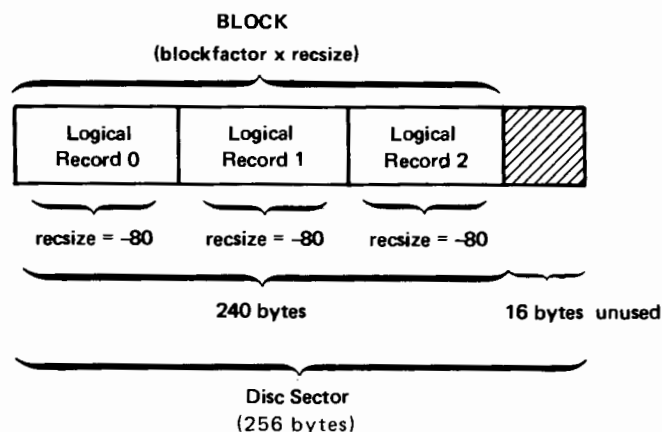
```
:FILE LISTREP, NEW; DEV = LP, , 4  ← Requests 4 copies
:RUN REPGEN
```

Note omitted output priority parameter

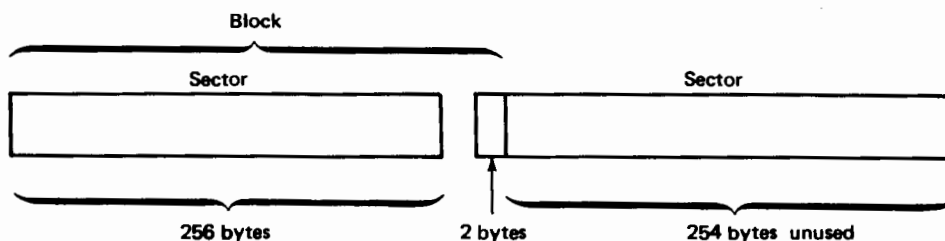
LOGICAL RECORDS AND BLOCKS

Programs can write logical records to a file in one of three formats: fixed-length, variable-length, and undefined length, as discussed below. If the file resides on disc, it may be organized for either sequential access (permitting fixed-length, variable-length, or undefined-length records) or for direct-access (allowing only fixed-length or undefined length records). If the file is a magnetic tape file, which is always organized for sequential access, the records may be written in any of the three formats. If the file is any other devicefile, MPE overrides any specification you supply, writing undefined-length records only.

FIXED-LENGTH RECORDS. On a file containing *fixed-length logical records*, all records are the same size. On disc and magnetic tape files, each block contains one or more logical records, but on files on other devices, blocks are equivalent to logical records. MPE determines the block size by multiplying two parameters supplied in the FOPEN intrinsic — the *logical record size* times the *blocking factor* (number of records per block). When written to disc, each block begins at the start of a sector. Thus, on a disc file, a 240-byte block containing three 80-byte fixed-length records would appear as follows:



When the next block is written to the disc file shown above, the last 16 bytes of the sector illustrated will remain unused. This occurs because, as a function of the disc controller, blocks always begin on sector boundaries. The controller can only transfer data in units equivalent to the length of a sector, 256 bytes. Thus, when MPE moves a 240-byte block from your program's stack through the controller to disc, the controller adds 16 bytes to equate the data transferred to the sector length. Because of this factor, you can waste disc space if you do not block your records judiciously. For instance, if you use a block factor of 1 when writing a fixed-length record of 258 bytes, you will waste 254 bytes of disc space as shown below. (When a block longer than 256 bytes is written, the last two data bytes are duplicated through the end of the following sector.)



For optimum use of disc space, compute the block size so that:

$$(\text{recsize} \times \text{blockfactor}) \text{ modulo } 256 = 0 \text{ (for bytes)}$$

or

$$(\text{recsize} \times \text{blockfactor}) \text{ modulo } 128 = 0 \text{ (for words)}$$

On magnetic tape files, you can best minimize waste of space by using large blocks containing several logical records; this reduces the total number of inter-block gaps on the tape.

Other examples of disc space used by fixed-length records appear in figure 6-6.

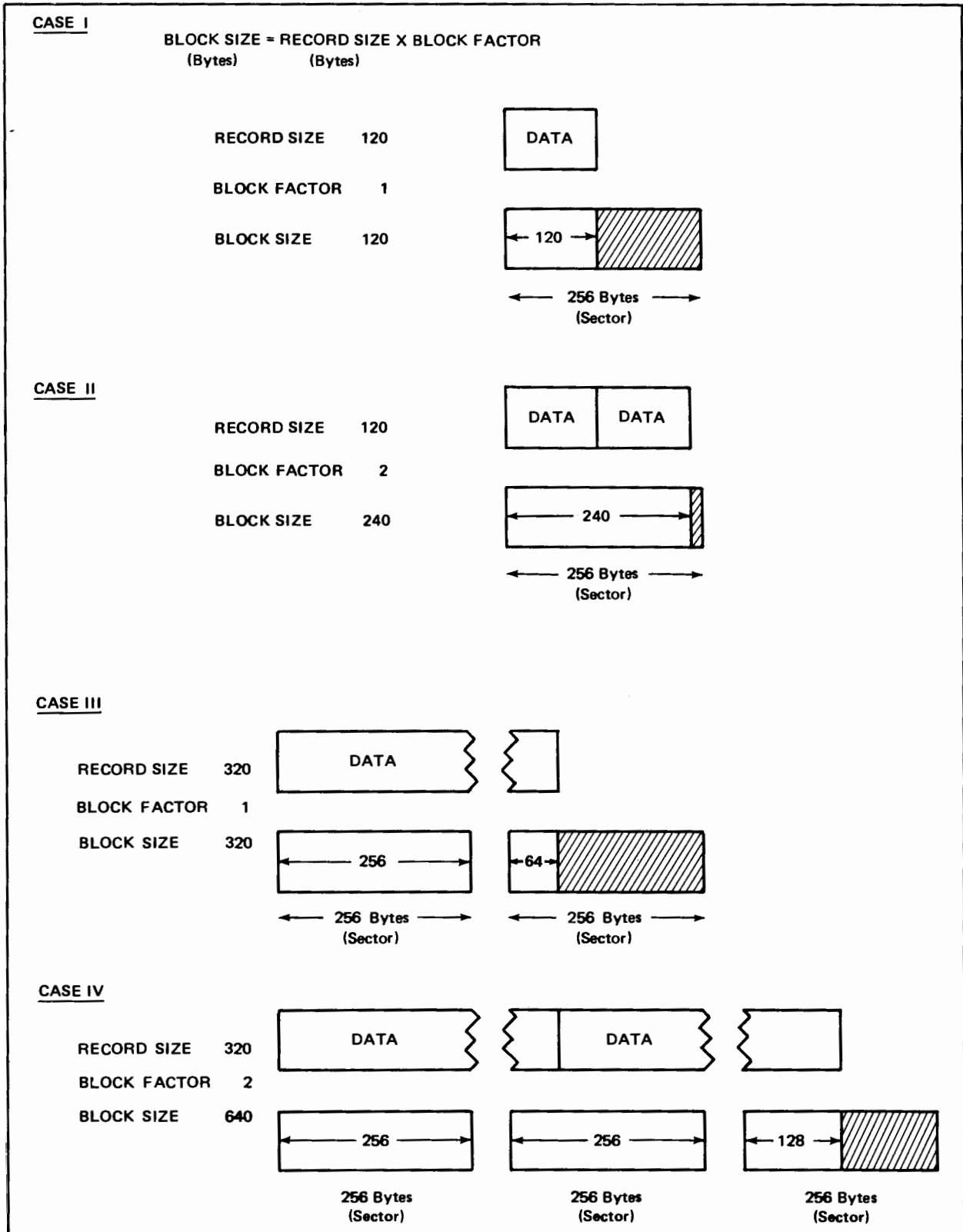
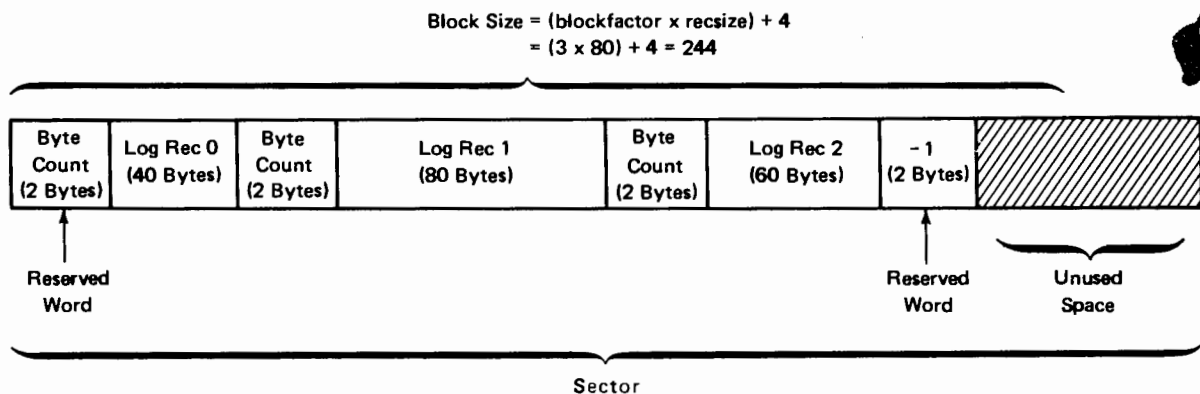


Figure 6-6. Disc Space Used by Fixed-Length Records

VARIABLE-LENGTH RECORDS. On a file containing variable-length records, records can vary in size with respect to each other. On both disc and magnetic tape files, each block contains one or more logical records. MPE determines the block size by multiplying the size of the longest record to be written times the blocking factor, as determined in FOPEN, and then adding four bytes reserved for use by the File Management System:

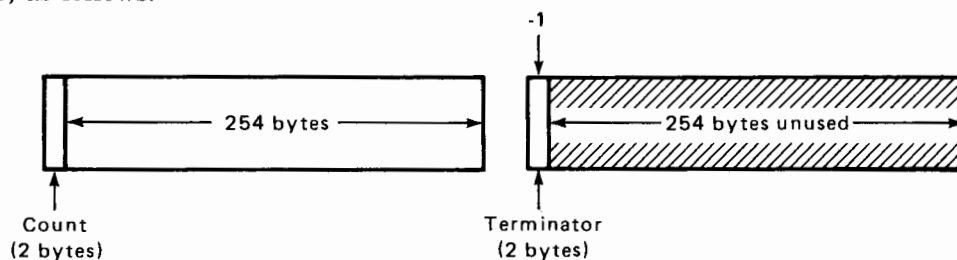
$$\text{Blocksize} = (\text{blockfactor} \times \text{recsize}) + 4 \text{ (Bytes)}$$

When a block is written to disc, each logical record is preceded by a one-word *byte-count* showing the length of that record in bytes. The last record in the block is followed by a word containing the characters - 1, acting as the block terminator; the next logical record in the file will be the *first* record in the next block. Of the four bytes reserved for system use when the blocksize is calculated, two comprise the word used for the byte count of the first record in the block, and two comprise the word used for the block terminator. On a disc file containing variable-length records, specified with a blockfactor of 3 and maximum record size of 80 bytes, a block might appear as follows:



You can determine the longest variable-length logical record that your program can write to a file by subtracting 4 from the block size (in bytes). Thus, for the file in the above example, the longest record possible is 240 bytes. Notice that the blocking factor applied to variable length records, unlike that applied to fixed-length records, does not specify the number of records per block - it is only a factor used to determine the block size. For instance, the block shown in the above example could easily accommodate six logical records of ten bytes apiece, each preceded by its own byte count, and still have much space left over.

As with fixed-length records, it is easy to waste disc space if you do not try to block your records efficiently. As an example, a record size of 256 and a blockfactor of 1 results in a waste of 254 bytes of disc space, as follows:



For best use of disc space, your blocksize should be an integral multiple of 256 bytes. Stated another way, the blockfactor should be computed so that:

$$(\text{recsize} \times \text{blockfactor}) \text{ modulo } 256 = 0$$

Additional examples of disc space usage by variable-length records appear in figure 6-7.

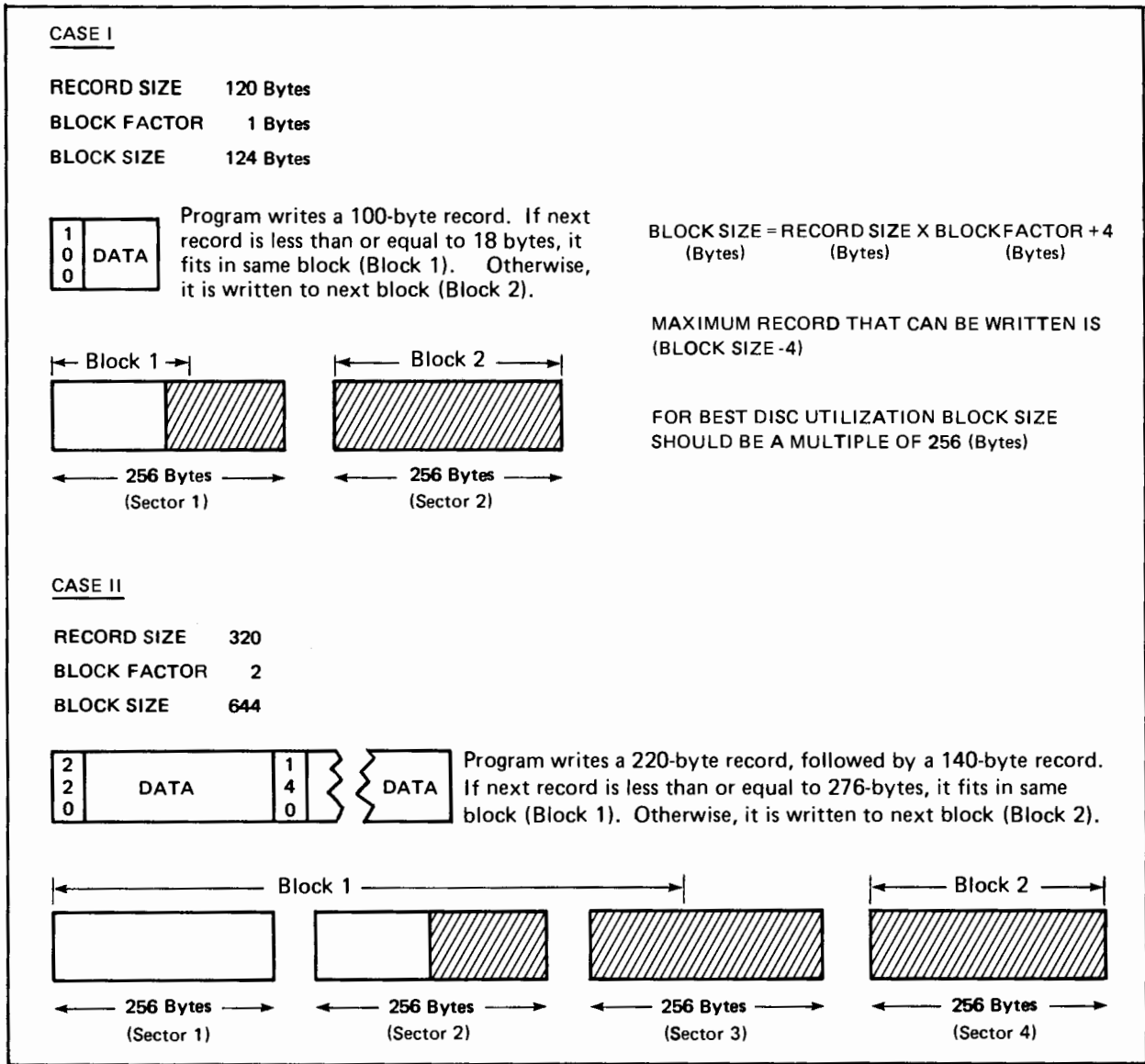
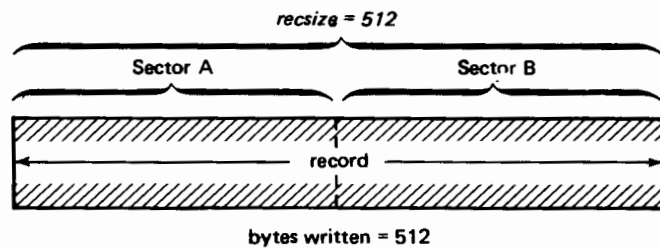


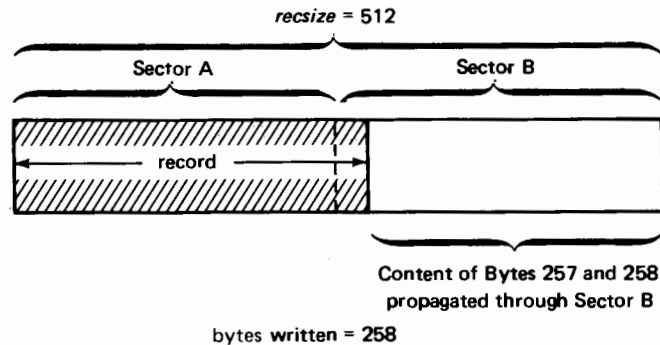
Figure 6-7. Disc Space Used by Variable-Length Records

UNDEFINED-LENGTH RECORDS. On a file containing undefined-length records, logical records and blocks are equivalent. The block factor is always 1. The record size parameter specified when you open the file denotes the length of the longest record to be transferred. Records of this type are written directly from stack to disc, with no buffers constructed for the file by MPE. The format of such records on disc, with respect to sectors occupied, can be illustrated by three cases in which the user-specified record size is 512 bytes:

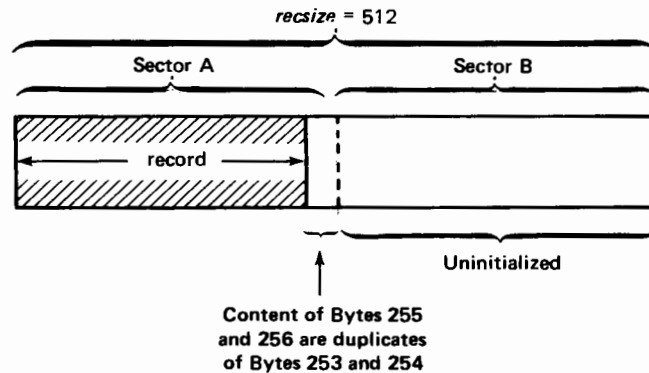
Case I : You write a record 512 bytes long. The full record/block completely fills two disc sectors.



Case II : You write a record 258 bytes long. The record written occupies all of Sector A and the first two bytes (word) of Sector B; the last two bytes written are duplicated throughout the remainder of Sector B. The rule is: if (record size) modulo 256 is not zero, then the last pair of bytes (word) written is duplicated through the last sector.



Case III: You write a record 254 bytes long. The record occupies 254 bytes of Sector A; the last pair of bytes (word) is duplicated throughout the remainder of Sector A. Sector B contains uninitialized data. The rule is: any sector not written into will remain uninitialized to 0 (binary files) or blanks (ASCII files).



DEVICE CONSTRAINTS ON RECORDS AND BLOCKS. When the System Supervisor or Console Operator adds a device to the system, he always specifies the default record size that applies to files residing on that device. For any device, this value determines the size of the logical records and blocks assigned if you do not specify a record size or blocking factor when you open a file on this device. The standard default record sizes for various devices are listed in table 6-6.

NOTE

The default values used by compilers and subsystems often differ from those used by MPE for specific devices. For instance, a text file saved on disc by the MPE Editor will have a default size of 80 or 72 bytes (depending on the text format), rather than the standard MPE default size of 256 bytes. Bear such differences in mind when using other subsystems.

Table 6-6. Standard Default Record Sizes

DEVICE	RECORD SIZE (BYTES)
Disc	256
Magnetic Tape Unit	256
Terminals (most cases)	80
Card Reader	80
Line Printer	132
Paper Tape Reader	80
Paper Tape Punch	256
Plotter	510
Printing Reader/Punch	No. of card columns, usually 80.
Programmable Controller	256
Synchronous Single-Line Controller	256

If you do not specify a record format or blockfactor in your FOPEN request, MPE assigns the format described below and a blockfactor of 1 to your file:

- *For Unit-Record Input Devices* (such as card readers) and *Unit-Record Output Devices* (such as lines printers), the default is undefined-length records.
- *For Magnetic Tape Units and Discs*, the default is fixed-length records.

ASCII VS. BINARY CODE. Programs can write records to a file in the American Standard Code for Information Interchange (ASCII) or in binary code. The code selected depends upon the purpose of the file. Text files of any kind are typically written in ASCII, where each byte equals one alphanumeric or special character. These files include those read from the standard input device, containing MPE commands, user programs, and/or data; source-program text files created by using the MPE Editor; and job listing files written to the standard listing device. On the other hand, many files created by MPE subsystems are written in binary code. These include user subprogram library (USL) files generated by a compiler or program files prepared for execution by the MPE Segmenter.

A program can specify the type of code used when it opens the file (or accept the MPE default, which depends on the device used for the file). At this time, you should also consider the best blocking factor to use for each type of file. For ASCII files, which MPE subsystems and user programs frequently write using 80-character records, 16 is generally a good blocking factor because it results in a block that exactly fills five disc sectors with no wasted space. For binary files, a good rule is to make the blocking factor some integral multiple of 256 bytes.

IMPROVING INPUT/OUTPUT EFFICIENCY. When you run a program that transfers data to or from a different input or output device from time to time, you can make the physical input or output more efficient by overriding the programmatically-specified record size or blocking factor so that these values better suit the device involved. For instance, suppose you are running a program originally written to read input from cards specified as 20-character logical records. If, before the next run, the input file has been copied to disc, you could provide faster access by reading these records in blocks of 240 characters. To do this, you would enter a :FILE command using a blocking factor of 12:

```
:FILE CARDS; DEV=DISC; REC=-20,12 ← Blockfactor
:RUN PROGX
```

↙ Resize

NOTE

When you specify record size in *bytes*, you must precede this value with a minus sign as shown above. When you express record size in words, however, be sure to omit the minus sign.

CONSERVING DISC SPACE. Suppose you are writing 80-byte records to a very large file on disc, and want to do everything possible to ensure that you do not run out of disc space for this file. You can do this by making sure that your blockfactor is some integral multiple of 256 bytes. In this case, a block factor of 16 is ideal because 80 bytes x 16 = 1280 bytes (exactly five disc sectors). You could specify this as shown below:

```
:FILE MYRECS; DEV=MHDISC; REC=-80, 16 ← Blockfactor
:RUN PROGR
```

↙ Resize

The constraints on disc space allowed for a file, and how that space is allocated, are covered in the discussion of disc extents on pages 6-42 through 6-44.

READING TAPES WITH UNKNOWN CONTENTS. If you have a magnetic tape whose contents (and record format) you do not know, you can determine its contents by copying it to a line-printer (via FCOPY) with a blockfactor of 1 and undefined-length record format. For instance,

```
:FILE TFILE; DEV=MT; REC=1024,1,U; BUF=1
:FILE PFILE; DEV=LP
:RUN FCOPY.PUB.SYS
>FROM=*TFILE; TO=*PFILE; OCTAL; CHAR; NORECNUM
```

↙ Resize is a number larger than the longest record you anticipate

Specifying undefined-length records is also useful when copying a tape that you know contains records of different sizes.

SUBMITTING JOBS AND DATA FROM MAGNETIC TAPE FILES. When MPE detects a :JOB command (to initiate a batch job) or a :DATA command (to spool data) on a magnetic tape file, it reads the commands or data from the tape using the configured default record size (*recsize*) of the tape unit and a blocking factor of one. You cannot alter this operation. Therefore, if you are creating a tape containing a job or :DATA information (via FCOPY), you should ensure that it contains one card image per block, as follows:

```

:FILE JOBTAPE; DEV=TAPE; REC=-80, 1, F, ASCII
:RUN FCOPY.PUB.SYS
>FROM=DISCFL; TO=*JOBTAPE }

```

recsize *blockfactor* *ASCII code*
Specifies tape output.

*Copies job from disc file to magnetic tape.
Disc file contains batch job created by
Editor, beginning with :JOB and termi-
nating with :EOJ.*

NOTE

The tape unit from which the job or data file is read must be configured to accept the required :JOB or :DATA commands.

PADDING RECORDS. Programs sometimes use files that must be initialized by filling them with either blank characters or zeros. Since unused space in ASCII files is always padded with blanks, and unused space in binary files is always filled with zeros, you can select the padding you desire by specifying the corresponding code. For example:

```

:FILE EMPTY, NEW; REC=-80, 1, F, BINARY
:RUN PROGX

```

*Creates temporary disc file
padded with zeros.*

Padding is only done up to the current end-of-file indicator. Padding also occurs within records if the written records are less than the specified logical record size and are written in fixed-length or undefined-length format.

INPUT/OUTPUT BUFFERS

If you do not specify otherwise when you open a file, MPE assigns the file two buffers into which it interleaves the transmission of data to and from the file. For instance, if your program is reading data from a sequential disc file in blocks of three records each, upon the first read request, MPE automatically moves the first three records from the file into the first buffer (Buffer 1) and the next three records into the second buffer (Buffer 2). When your program has read all three records from Buffer 1 and accesses the first record in Buffer 2, MPE automatically moves the *next* three unread records in the file into Buffer 1 so that they will be immediately available for any upcoming read request; when your program reads all records in Buffer 2, MPE moves another three records into Buffer 2, continuing in this fashion until the program terminates access to the file. This technique, known as *anticipatory reading*, effectively permits the overlapping of input/output requests, often significantly reducing the time required to process a file. It is most effective in purely sequential-access operations but can also be used in conjunction with the FREADSEEK intrinsic when you wish to access records non-sequentially.

Typically, two buffers are sufficient for most applications, but occasionally additional buffers can improve your access time. This is most likely in cases where few sessions/jobs are running on the system, and your application processes each record rapidly but reads the records from a slow input/output device — for instance, when you are using FCOPY to copy a file from cards to disc. A useful guideline is to try your program with the default of two buffers and if you find you require faster input/output, then advance to three buffers. Specify the additional buffer as shown in the example below:

```
:FILE CARDS; DEV=CARD; REC=-80, 3; BUF=3 ← Specifies 3 buffers.
:RUN FCOPY.PUB.SYS
>FROM=*CARDS; NEW; TO=DISKFL
>EXIT
```

NOTE

The maximum number of buffers you can specify is 16. However, any number beyond 3 does not usually increase input/output efficiency and needlessly occupies space in main memory.

If you specify BUF=0, MPE overrides this and supplies the standard default of 2.

For files input or output at interactive terminals, you need not specify *any* buffer parameter; a system-managed buffering operation is always used for terminals. (If you do specify any buffers for a terminal, MPE overrides this specification and assigns no buffers.)

Files with variable-length records always require buffering.

The maximum total buffer space available for an individual file is 8,192 words (16,384 bytes). Thus, in no case will a file access be granted buffer space (block size x number of buffers) in excess of this value.

On occasion, you may wish to avoid the use of buffers altogether. This may be the case, for instance, where you are transferring records in large blocks. (These can require excessive amounts of memory for the data transferred and additional overhead for pointers and file-access information required to maintain the buffers.) Furthermore, certain file-access modes prohibit the use of buffers; for example, multi-record (MR) access and NOWAIT input/output, discussed later in this section, are incompatible with buffering. (If you request buffering in such cases, MPE overrides your request and allocates no buffers.) To expressly specify no buffering, enter the NOBUF keyword parameter in the :FILE command, as follows:

```
:FILE BIGDATA; REC = -4096, 16, F; NOBUF ← Specifies no buffers.
```

MULTI-RECORD MODE

In almost all applications, programs conduct input/output in normal recording mode, where each read or write request transfers one logical record to or from the data stack. In certain cases, however, you may want your program to read or write, in a single operation, data that exceeds the logical record length defined for the input or output file — for instance, you may want to read four 128-byte logical records from a file to your stack in a single 512-byte data transfer. Such cases primarily arise in specialized applications. Suppose, for example, that your program must read input from a disc file containing 256-byte records. This data, however, is organized as units of information that may range up to 1024 bytes long; in other words, the data units are not confined to record boundaries. Your program is to read these units and map them to an output file, also containing 256-byte records. You can bypass the normal record-by-record input/output, instead receiving data transfers of 1024 bytes each, by specifying the multi-record (MR) mode in your FOPEN call or :FILE command. For example,

```
:FILE BIGCHUNK; REC=-256, 1, U; NOBUF; MR
```

Specifies multi-record mode ↗

When used, multi-record mode is primarily applied to disc files, although it is not confined to them. Because this mode effectively ignores logical record boundaries, it also implicitly ignores block boundaries. Multi-record mode requires no buffers and a blockfactor of 1, and MPE assigns these characteristics even if you specify otherwise in your FOPEN request or :FILE command.

RESTRICTING FILE ACCESS (BY ACCESS-MODE)

When a program creates a file, it can restrict access to the file by specifying a particular access mode (such as read-only, write-only, update, and so forth) for the file. These restrictions apply to files on any device, and can be changed or overridden only by yourself (as the creator of the file). They are discussed below. In addition, for files on disc, a program can also restrict access so that only one access-attempt (FOPEN call) or process (running program) can open it at one time, or can allow it to be shared among several accesses. These restrictions are described under *Disc File Considerations* on page 6-40. The specifications established for a file when it is created may be overridden at the account and group level by other restrictions imposed by the MPE File Security System; those restrictions are discussed under *Specifying File Security* on page 6-74.

The access types that can be specified by a program are listed in table 6-7.

Table 6-7. File Access Types

ACCESS-MODE	:FILE PARAMETER	DESCRIPTION
Read-Only	IN	Permits file to be read but not written upon. Used for devicefiles such as card reader and paper-tape reader files, as well as magnetic tape, disc, and terminal input files.
Write-Only	OUT	Permits file to be written upon but not read. Any data already on the file is <i>deleted</i> when the file is opened. Used for devicefiles such as card punch, line printer, as well as tape, disc, and terminal output files.
Write (Save) Only	OUTKEEP	Permits file to be written upon but not read, allowing you to add new records both before and after current end-of-file indicator.
Append Only	APPEND	Permits information to be appended to file, but does not allow over-writing of current information nor reading of file. Allows you to add new records after current end-of-file indicator only. Used when present contents of file <i>must</i> be preserved.
Input/Output	INOUT	Permits unrestricted input and output access of file; information already on the file is saved when file is opened. (In general, combines features of IN and OUTKEEP.)
Update	UPDATE	Permits records in file to be updated—record is read into your data stack, altered, and re-written to file. All data already in file is saved when file is opened.

When specifying the access mode for a file, it is important to realize where the current end-of-file is before and after the file is opened and where the logical record pointer indicates that the next operation will begin. These factors depend upon the access mode you select. Because they are best explained by example, the effects of each access mode upon these factors are summarized in table 6-8 for a file that contains ten logical records of data (numbered from 0 through 9). This table shows that the current end-of-file (EOF) lies at Record 10 before the file is opened, indicating that if another record were appended to the file, that would be the eleventh record. When you open the file in the write-only mode, however, all records presently in the file are deleted and the logical record pointer and current EOF move to Record 0. Now when you write a record to the file, this will be the first record in that file.

Table 6-8. Effects of Access Modes

ACCESS MODE	CURRENT EOF	LOGICAL RECORD POINTER	EOF AFTER OPEN
Read only	10	0	10
Write only	10	0	0
Write (save) only	10	0	10
Append	10	10	10
Input/Output	10	0	10
Update	10	0	10
Execute	10	0	10

When you are running a program that opens a magnetic tape file for write-only access but wish to append records to that file rather than delete existing records, you can override the programmatic specifications by using the :FILE command to open the file for append-access, as follows:

```
:FILE TASK; DEV=MT; ACC=APPEND ← Requests append access
:RUN PROGM
```

Suppose you run a program that opens a disc file for write-only access, copies records into it, and closes it as a permanent file. Under the standard MPE security provisions, the access mode is automatically altered so that the file permits the read, write, and append access modes (among others). Now, suppose you run the program a second time, but wish to correct some of the data in the file rather than delete it. You could use the :FILE command to override the programmatic specification, opening the file for update access:

```
:FILE REFILE; ACC=UPDATE ← Requests update access
:RUN PROGN
```

Consider a program that reads input from a terminal (file name INDEV) and directs output to a line printer (OUTDEV). You can re-direct the output so that it is also transmitted to the terminal by entering:

```
:FILE INDEV; DEV=TERM; ACC=INOUT Re-specifies INDEV for both input and output access.
:FILE OUTDEV=*INDEV Equates OUTDEV to INDEV.
:RUN PROGO Runs program.
```

CARRIAGE-CONTROL CHARACTERS

Programs that write output to new ASCII files on line printers or terminals can direct carriage spacing and skipping via carriage-control characters. These characters each indicate a unique carriage operation. For example, the control character octal 60 (%60) results in double-spacing the carriage. A program can transmit control characters in either of two ways, specified in the FOPEN intrinsic:

1. Embedded as the first character of each data record transmitted by an FWRITE request, or

2. Specified as the *control* parameter of the FWRITE request that writes each record.

If the program does not explicitly specify which of these options takes effect when it opens a file on a device having a print carriage, MPE automatically adds an extra byte via the File Management System that is used by the device driver for carriage control.

Some MPE subsystems as well as many user programs open files under the MPE default where carriage-control characters are expected as part of the data. You may wish to override this specification via the :FILE command in certain cases. In fact, when running a FORTRAN program that references the file designator FTN06 (the standard list device), you must do this because FORTRAN supplies its own carriage-control characters with each write request but does not specify this option when it opens the file. Thus, to transmit data to a line printer when running a FORTRAN program, you might enter:

Denotes control-character transmitted by FWRITE parameter. ↘

```
:FILE FTN06; DEV=LP; REC=-132, 1, V, ASCII; CCTL  
:RUN FPROG ← Runs FORTRAN program.
```

When a carriage-control character is sent to a device on which carriage control cannot be executed directly, such as disc or tape, that character is embedded as the first byte of the record transmitted.

When you transmit a carriage control character in the FWRITE call, this adds an additional byte to the record size when the file is created. Thus, the record size of 132 bytes specified in the previous example actually results in records 133 bytes long.

NOTE

You can only specify CCTL in the :FILE command for devicefiles or new disc files written in ASCII code.

For further information about specific control characters and their effects, as well as the FWRITE command and its parameters, see *MPE Intrinsic Reference Manual*.

FILE CODES

MPE subsystems often create special-purpose files whose functions are identified by four-digit integers called file codes, written in their labels. For instance, compilers create user-subprogram library (USL) files, written in a special format and identified by the code 1024, upon which they compile object programs. User programs sometimes create files that must be identified in some unique way, too. Such a program might produce a permanent disc file identified by the integer 1. If you were to run this program several times and wanted to uniquely identify the file produced on each run (or set of runs) by a special class, purpose, or function, you could use a :FILE command to supply a unique file code for each run (or group of runs). For instance, on the second run, you might wish to classify the file with the file code 2, as follows:

```
:FILE DESGX=DESGB; CODE=2 ← File code  
:RUN FILEPROD
```


- Whether the file is restricted to one access or may be shared among several simultaneous accesses or running programs.
- Whether the file is to be saved in or deleted from the system when it is closed.

DETERMINING FILE SPACE REQUIRED. When a program opens a new file on disc, it specifies the maximum capacity of the file in terms of logical records (for files with fixed-length records) or blocks (for files with variable- or undefined-length records). If the program does not explicitly specify the capacity, MPE assigns a default value of 1023 for this parameter; thus, a file containing fixed-length records would contain a maximum of 1023 records; a file containing variable-length records could hold a maximum of 1023 blocks. Regardless of the type of records written to the file, the total disc space allocated cannot exceed 2,097,152 sectors. When a file is defined for fixed-length records, you can calculate the number of sectors that it will require by using this formula:

$$S = \left\lceil \frac{N}{B} \right\rceil \times \left\lceil \frac{L \times B}{256} \right\rceil$$

<i>S</i>	The number of disc sectors required by the file.
<i>N</i>	The number of fixed-length logical records in the file.
<i>B</i>	The number of logical records in a block (blocking factor).
<i>L</i>	The length of each logical record, in 8-bit <i>bytes</i> .

This formula applies to every type of disc drive supported by MPE. In this formula, the constant 256 is the number of bytes in a sector. Each expression within brackets is evaluated separately and rounded upward before the final multiplication takes place. (A notation in the form $\lceil x \rceil$ means “ceiling (x)” —the smallest integer greater than or equal to x.) The formula does not include disc space required for the file label. To illustrate the use of the formula, suppose you wished to calculate the effective disc space required for a file to contain 100 records of 100 bytes (50 words) each, established with a blocking factor of 3. You would proceed as follows:

$$\begin{aligned} S &= \left\lceil \frac{100}{3} \right\rceil \times \left\lceil \frac{100 \times 3}{256} \right\rceil \\ &= \lceil 34 \rceil \times \lceil 2 \rceil \\ &= 68 \text{ sectors} \end{aligned}$$

Suppose that you wish to run a program that writes 100-byte fixed-length records to a new file with a blockfactor of three, and opens the file for a maximum capacity of 100 records as in the above example. On this run, however, you wish to open the file for a greater capacity, say 200 records. You could override the programmatic specification and request a file of this length by entering the `:FILE` command as follows:

```
:FILE DISKFILE; DISC=200 ← Specifies file capacity
:RUN PROGX
```

The resulting file would require 134 sectors, calculated as follows:

$$\begin{aligned}
 S &= \left\lceil \frac{200}{3} \right\rceil \times \left\lceil \frac{100 \times 3}{256} \right\rceil \\
 &= [67] \times [2] \\
 &= 134 \text{ sectors}
 \end{aligned}$$

You can only create a new disc file under your log-on account. If you plan to make this a permanent file, you may be constrained by limits on the amount of disc space available for permanent files (established at the account level by the System Manager and at the group level by the Account Manager). The limits are established in terms of disc sectors. When someone attempts to save a new disc file or to create or add sectors to a permanent file, the request is denied if it would exceed one of these limits. (When the MPE default accounting provisions are in effect, however, permanent file space accumulated by users is monitored but not limited.) If you suspect that you will encounter a limit, you can determine the amount of file space available (in sectors) by entering the :REPORT command. The report generated by this command indicates the maximum space originally available for the account and each of its groups, and how much has been used as of this moment. It also provides statistics for central-processor and connect time available and used. The report shown below indicates that under the account TECHPUBS, and log-on group XGROUP, 182 of the 5000 sectors available have been used.

```

:REPORT
ACCOUNT          FILESPACE-SECTORS      CPU-SECONDS      CONNECT-MINUTES
  /GROUP          COUNT      LIMIT      COUNT      LIMIT      COUNT      LIMIT
TECHPUBS          182      5000      8          **          5          **
  /XGROUP          182      5000      8          **          0          **
  
```

If you see that a limit is about to be reached, contact your System or Account Manager about raising the existing limits or clearing the resource-use counters.

ALLOCATING FILE SPACE. MPE manages each file on disc as a set of extents; each *extent* is an integral number of consecutively-located disc sectors. A program can specify the number of extents for a file in the FOPEN intrinsic or accept the MPE default of eight. All extents (except possibly the last) are equal in size. You can determine the size of each extent by the following formula:

$$S_e = \left\lceil \frac{N}{B \times E} \right\rceil \times \left\lceil \frac{B \times L}{256} \right\rceil$$

- S_e The number of disc sectors in the extent.
- N The number of fixed-length logical records in the file.
- B The number of logical records in a block (blocking factor).
- E The number of extents in the file.
- L The length of each logical record, in eight-bit bytes.

In this formula, the constant 256 denotes the size of each sector in bytes. In some cases, the last extent is smaller than the others.

To illustrate the use of the formula, the extent size is calculated below for a file containing 1024 logical records, organized as eight extents, with a blockfactor of 3. Each record is an 80-byte card image. The extent size is:

$$\begin{aligned} S &= \left\lceil \frac{1024}{3 \times 8} \right\rceil \times \left\lceil \frac{3 \times 80}{256} \right\rceil \\ &= 43 \times 1 \\ &= 43 \text{ sectors.} \end{aligned}$$

(The first seven extents would each contain 43 sectors and the last extent would contain 41.)

With a blockfactor of 16 applied in the above example, the extent size is:

$$\begin{aligned} S_e &= \left\lceil \frac{1024}{16 \times 8} \right\rceil \times \left\lceil \frac{16 \times 80}{256} \right\rceil \\ &= 8 \times 5 \\ &= 40 \end{aligned}$$

In this case, all extents contain 40 sectors.

When a file is opened, the first extent (containing at least one sector for the file label) is allocated immediately; other extents up to a maximum of 32 are allocated as they are needed. Alternatively, you can request immediate allocation of more than one extent when the file is opened. This ability to divide a file into extents and to specify how much of it is allocated and when, enables you to optimize file access to save disc space, as described below.

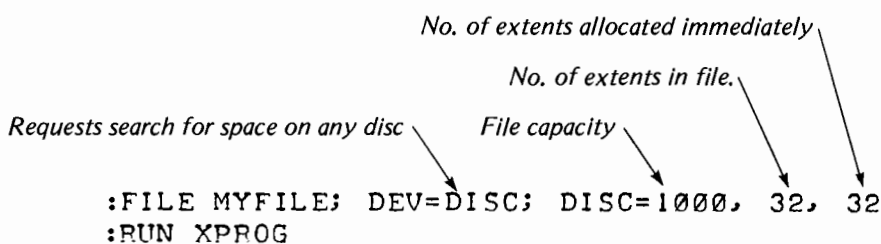
The extents comprising a file are restricted to discs that belong to the device class specified when the file was created. Normally, each extent is arbitrarily assigned to any device within this class. Alternatively, you can restrict all extents to the same disc device by identifying the file by a logical unit number rather than a device class name when you open it. In any case, MPE always ensures that each individual extent resides in total on the same disc device and is not spread over two or more devices. (If your system contains two or more discs with class name DISC, and they are different in speed, you may wish to select a particular disc for a file on the basis of the amount of access expected for that file.)

NOTE

Spooled devicefiles (spool files) are written in a special format and managed entirely by the File Management System. They are limited to a maximum of 32 extents; the size of their extents is determined by the System Manager when he configures the system.

Occasionally, you may wish to override programmatic specifications dealing with extents to optimize the use of disc space. For instance, if your disc space is limited, the space available may exist as isolated small groups of sectors (fragments) rather than as contiguous groups of many sectors. You may then decide to "break" the file into more extents, each small enough to fit into the fragments

available. If you fail to do this, perhaps attempting to open the file with one extent, you may not get the disc space you require. To illustrate, if your disc space is limited and you run a program to create a new file for 1000 records of 80 bytes, treated as 10 extents, you could enhance your chances of acquiring the disc space needed by issuing a :FILE command specifying 32 extents, all allocated immediately:



In general, the rule is: *if your disc space is limited and you know the total space your file will need, divide the file into many extents and request immediate allocation of all of them.*

Extents are allocated and initialized by filling them with blanks (ASCII files) or zeros (binary files) the moment a program attempts to read or write on them for the first time. Because they are allocated at these times, you must use caution when performing direct access of disc files (via the FREADDIR or FWRITEDIR intrinsic). For instance, suppose you open a new file containing eight extents and immediately allocate the first of these. You then, via direct access, allocate and write data into the fifth extent, and close the file. The next time you run your program, disc space may be limited so that no further extents of the size required can be allocated your file. In this case, if you open the file and try to write data into the second, third, or fourth extent, the attempt fails because these extents cannot be allocated.

Some files used by MPE command executors are restricted to only one extent. For instance, a program file prepared by the MPE Segmenter falls into this category. If you are preparing a program onto an existing file in the session/job temporary domain, you can ensure that the file is limited to one extent by specifying the :FILE command as shown in this example:

```

:FILE PROGFL; DISC= , 1; CODE=PROG    Specifies file of one extent.
:PREP COMFILE, PROGFL                Prepares program onto file PROGFL.
  
```

If you violate the single-extent rule for a subsystem or command executor that requires it, you will receive the following message:

MORE THAN ONE EXTENT IN PROGRAM

SIMULTANEOUS ACCESS OF FILES. When an FOPEN request is issued for a file, that request is regarded as an individual accessor of the file and a unique file number, set of buffers, and other file control information is established for that file. Even when the same program issues several different FOPEN calls for the same file, each call is treated as a separate accessor. Under the normal (default) security provisions of MPE, when an accessor opens a disc file not presently in use, the access-restrictions that apply to this file for other accessors depend upon the access-mode requested by this initial accessor:

- If the first accessor opens the file for read-only access, any other accessor can open it for any other type of access (such as write-only or append), except that other accessors are prohibited exclusive access.

- If the first accessor opens the file for any other access mode (such as write-only, append, or update), this accessor maintains exclusive access to the file until it closes the file; no other accessor can access the file in any mode.

Programs can override these defaults by specifying other options in FOPEN intrinsic calls. Users running those programs can, in turn, override both the defaults and programmatic options through the :FILE command. The options are listed in table 6-9. The actions taken by MPE when these options are in effect and simultaneous access is attempted by other FOPEN calls are summarized in table 6-10. (This table also denotes the effects of simultaneous access attempts via other MPE operations and commands described later in this manual.) The action taken depends upon the current use of the file versus the access requested.

Table 6-9. File-Sharing Restriction Options

ACCESS RESTRICTION	:FILE PARAMETER	DESCRIPTION
Exclusive Access	EXC	After file is opened, prohibits concurrent access in <i>any</i> mode through another FOPEN request, whether issued by this or another program until this program issues FCLOSE or terminates.
Exclusive Write Access	EAR	After file is opened, prohibits concurrent write access through another FOPEN request, whether issued by this or another program, until this program issues FCLOSE or terminates.
Sharable Access	SHR	After file is opened, permits concurrent access to file in any mode through another FOPEN request issued by this or another program, <i>in this or any other session or job</i> . Each accessor uses <i>copy</i> of portion of file within its own buffer.
Multi Access	MULTI	After file is opened, permits concurrent access to file in any mode through another FOPEN request issued by this or another concurrently running program in <i>this</i> session or job. Each accessor uses same buffers and access-control information, with blockfactor specified by <i>this</i> initial accessor. Information is transferred to and from the file in the order it is requested, regard less of which accessor requests the transfer.

REQUESTED ACCESS GRANTED, UNLESS NOTED

Requested Access	Current Use	FOPEN for Input		FOPEN for Output		FOPEN for Input/Output		Program File Loaded	Being :STORED	Being :RESTORED
		SHR/MULTI	EAR	SHR/MULTI	EAR	SHR/MULTI	EAR			
FOPEN for Input	SHR/MULTI	Requested Access Granted	Requested Access Granted	Requested Access Granted	Requested Access Granted	Requested Access Granted	Requested Access Granted	Requested Access Granted	Requested Access Granted	Error Message
	EAR	Requested Access Granted	Requested Access Granted	Error Message	Error Message	Error Message	Error Message	Error Message	Error Message	Error Message
FOPEN for Output	SHR/MULTI	Requested Access Granted	Error Message	Requested Access Granted	Error Message	Requested Access Granted	Error Message	Error Message	Error Message	Error Message
	EAR	Requested Access Granted	Error Message	Error Message	Error Message	Error Message	Error Message	Error Message	Error Message	Error Message
FOPEN for Input/Output	SHR/MULTI	Requested Access Granted	Input Granted	Requested Access Granted	Input Granted	Requested Access Granted	Input Granted	Input Granted	Input Granted	Error Message
	EAR	Requested Access Granted	Input Granted	Error Message	Error Message	Error Message	Error Message	Error Message	Error Message	Error Message
: RUN, COMMAND, CREATE INTRINSIC		Requested Access Granted	Requested Access Granted	Error Message	Error Message	Error Message	Error Message	Requested Access Granted	Only if Loaded	Error Message
:STORE		Requested Access Granted	Requested Access Granted	Error Message	Error Message	Error Message	Error Message	Requested Access Granted	Error Message	Error Message
:RESTORE		Error Message	Error Message	Error Message	Error Message	Error Message	Error Message	Error Message	Error Message	Error Message

- NOTES: 1. SHR = Share; EAR = Exclusive, write-access; MULTI = Share file, data segment, and access-control information.
 2. Fully exclusive accesses cause any succeeding access (except :STORE) to fail.
 3. Append access treated like output; Update treated like input/output.

Table 6-10. Actions Resulting from Multi-Access of Files

Exclusive Access. This option is useful when you wish to update a file, and wish to prevent other users or programs from reading or writing on this file when you are using it. Thus, no user can read information that is about to be changed, nor can he alter that information. To override the programmatic option under which the file would be opened and request exclusive access, you could use the EXC keyword parameter in the :FILE command:

```
:FILE DATALIST; EXC
:RUN FLUPDATE
```

Requests exclusive access

Exclusive (Write) Access. This option allows other accessors to read the file but prevents them from altering it. When appending new part numbers to a file containing a parts list, for instance, you might use this option to allow other users to read the current part numbers at the same time you are adding new ones to the end of the file. You could request this option as follows:

```
:FILE PARTSLIST; EAR
:RUN FLAPPEND
```

Requests exclusive-write access



Share Access. When opened with the share option, a file can be shared (in all access modes) among several FOPEN requests, whether they are issued from the same program, different programs within the same session/job, or programs running under different sessions/jobs. Each accessor transfers its input/output to and from the file via its own unique buffer, using its own set of file control information and specifying its own buffer size and blocking factor. Effectively, each accessor accesses its own copy of that portion of the file presently in its buffer. Thus, share-access is useful for allowing several users to read different parts of the same file. It can, however, present problems when several users try to write to the file. For instance, if two users are updating a file concurrently, one could easily overwrite the other's changes when the buffer content from the first user's output is overwritten on the file by the buffer content from the second user's output. To use write-access most effectively with shared files, specify the multi-access option as discussed below. (The primary distinction between shared and multi-access is described in that discussion.)

To request share access for a file, use the SHR parameter in the :FILE command, as follows:

```
:FILE RDFILE; SHR
:RUN RDPROG
```

Requests shared access

Multi-Access. This option extends the features of the share-access option to allow a deeper level of multiple-access — it not only makes the file available simultaneously to other accessors (in the same session/job), but permits them to use the same buffers, blocking factor, and other file-control information. Thus, transfers to and from the file occur in the order they are requested, regardless of which program in your session/job does the requesting. When several concurrently-running programs (processes) are writing to the file, the effect on the file is the same as though one program was performing all output — truly sequential access by several concurrently-running programs.

NOTE

Multi-access allows the file to be shared (in all access modes) among several FOPEN requests from the same program, or from different concurrently-running programs in the same session/job. Unlike *share access*, however, *multi-access* does not permit the file to be shared between different sessions and jobs.

For further information on concurrently-running programs (processes), see *MPE Intrinsic Reference Manual*.

When the file is opened by the first accessor, MPE establishes the file specifications via the customary hierarchy (file label, :FILE command, FOPEN parameters, and MPE defaults in that order). These specifications (including buffering, blockfactor, and so forth) then apply to *all* other accessors (regardless of what they request) until the file is closed by all accessors. This allows many accessors to interleave their accesses of the file, actually transmitting data in the same order it is requested.

To illustrate the distinction between the share-access and multi-access options, suppose you were running a job executing three programs — A, B, and C — that sequentially access a disc file in read mode with a blockfactor of 3, via a buffer. Under the *share* option, the file would be read as follows. (In this example, records are indicated in the format *b-r*, where *b* is the number of the block and *r* is the sequential number of the record in the block.)

Program A reads records	1-1, 1-2, 1-3, 2-1, . . .
Program B reads records	1-1, 1-2, 1-3, 2-1, . . .
Program C reads records	1-1, 1-2, 1-3, 2-1, . . .

With *multi-access* mode, however, the running programs collectively work to perform true sequential access, eliminating the duplicate reading of records:

Program A reads records	1-1,	2-1,	3-1, . . .
Program B reads records	1-2,	2-2,	3-2, . . .
Program C reads records	1-3,	2-3,	3-3, . . .

As another example, suppose two programs running within a job sequentially access a serial device in the write-access mode, via a buffer with a blockfactor of 3. With the *share* option, the file would be written as follows:

Program A writes records	1-1, 1-2, 1-3, 3-1, 3-2, 3-3, . . .
Program B writes records	2-1, 2-2, 2-3, 4-1, 4-2, 4-3, . . .

With *multi-access* mode, the same file is written as follows:

Program A writes records	1-1,	1-3,	2-2,	3-1, . . .
Program B writes records	1-2,	2-1,	2-3,	3-2, . . .

You might wish to use the multi-access mode to allow several concurrently-running programs to update the same file. You could specify this option in the :FILE command as follows:

:FILE UFL; MULTI	<i>Specifies multi-access option for file named UFL.</i>
:RUN UPDPROG	<i>Runs program that creates and runs other programs (descendant processes), all operating on file named UFL.</i>

CLOSING DISPOSITION

As noted earlier, when a program closes a disc file, it can save the file in the system file domain or the session/job temporary file domain, or it can delete the file. Occasionally, you may wish to override the disposition specified in your program by entering one of the following parameters in the :FILE command: *DEL* (for delete), *SAVE* (for system file domain), or *TEMP* (for session/job temporary file domain). For instance, if you have a program that opens a file for scratch purposes and then deletes that file, but you wish on the next run to save the file (in the system file domain) to aid in debugging, you could enter:

```
:FILE SCRATCH; SAVE ← Saves file in System File Domain
:RUN PROG C
```

:FILE/FOPEN PARAMETER RELATIONS

Most file characteristics specified in the FOPEN intrinsic can be overridden via the :FILE command. A list of these characteristics, their related FOPEN parameters, and the corresponding :FILE command parameters appears in table 6-11. (Refer to *MPE Ininsics Reference Manual* for further details.)

The following example demonstrates the flexibility and power of the :FILE command in overriding many programmatic specifications for a file. Suppose that you have two FORTRAN programs that you want to execute within one batch job. Before the first program is executed, you must override its programmatic specifications for a new file named FDATA1 by supplying the following new specifications: a binary disc file, direct-access, with fixed-length records of 100 words (200 bytes) each. Maximum file size is 5000 records divided into 10 extents, but only one extent (500 records) is to be allocated initially. The initial FORTRAN program expects this file to be accessed as logical unit number 8. You wish to save this file in the system file domain. To do all this, you enter the following :FILE command and run the program (MYPROG1):

```
:FILE FTN08=FDATA1, NEW; SAVE; REC=100,, F, BINARY; DISC=5000, 10,1
:RUN MYPROG1
```

The file is created when opened by the FORTRAN program; it is permanently saved under your log-on group when closed by the FORTRAN program. After the first program is executed, you must use the :FILE command again because the second program expects to access this file through logical unit numbers 20 and 21. Rather than re-enter the above lengthy :FILE command, you simply enter the following :FILE commands and then run the program (MYPROG2).

```
:FILE FTN20=FDATA1
:FILE FTN21=*FTN20          (OR :FILE FTN21=FDATA1)
:RUN MYPROG2
```

Table 6-11. :FILE vs. FOPEN Parameters

CHARACTERISTIC	:FILE PARAMETER	FOPEN PARAMETER	MPE DEFAULT
Formal file designator.	<i>formaldesignator</i>	<i>formaldesignator</i>	Temporary nameless file.
Actual file designator.	<i>filereference</i> \$NEWPASS \$OLDPASS \$NULL \$STDIN \$STDINX \$STDLIST	Default file designator <i>foption</i> (Bits 10:3)	Same as formal file designator.
Domain	NEW OLD OLDTEMP	Domain <i>foption</i> (Bits 14:2)	New file.
Logical record size	<i>recsize</i>	<i>recsize</i>	Configured default size of device for unit-record devices; 256 bytes for other devices.
Block/buffer size	<i>blockfactor</i>	<i>blockfactor</i>	Configured block size of device divided by <i>recsize</i> .
Record format	F V U	Record format <i>foption</i> (Bits 8:2)	Fixed-length records for disc and magnetic tape files; undefined-length records for all others.
ASCII/Binary Code	ASCII BINARY	ASCII/Binary <i>foption</i> (Bits 13:1)	Binary.
Carriage-control characters supplied in FWRITE	CCTL NOCCTL	Carriage-control <i>foption</i> (Bits 7:1)	No carriage control characters supplied in FWRITE.
Access mode	IN OUT OUTKEEP APPEND INOUT UPDATE	Access-type <i>aoption</i> (Bits 12:4)	Read-only access for all devices except output devices, which are assigned output-only access.
Number of buffers	<i>numbuffers</i> NOBUF	<i>numbuffers</i> (Bits 11:5)	2 buffers.
Exclusive/Share access	EXC EAR SHR	Exclusive access <i>aoption</i> (Bits 8:2)	For read-only access, SHR takes effect; for other modes, EXC.
Multi access	MULTI NOMULTI	Multi-access mode <i>aoption</i> (Bits 6:1)	No multi access allowed.
Multi-record mode	MR NOMR	Multi-record <i>aoption</i> (Bits 11:1)	No multi-record mode.

Table 6-11. :FILE vs. FOPEN Parameters (Continued)

CHARACTERISTIC	:FILE PARAMETER	FOPEN PARAMETER	MPE DEFAULT
File disposition	DEL SAVE TEMP	(None—defined programmatically by <i>disposition</i> parameter of FCLOSE)	Same as when file was opened.
Device Class Name or Logical Device Number	<i>device</i>	<i>device</i>	Class Name DISC.
Output priority	<i>outputpriority</i>	<i>numbuffers</i> (Bits 0:4).	8
NOWAIT input/output	NOWAIT WAIT	NOWAIT I/O <i>aoption</i> (Bits 4:1)	NOWAIT input/output prohibited.
Number of copies	<i>numcopies</i>	<i>numbuffers</i> (Bits 4:7)	1
File code	<i>filecode</i>	<i>filecode</i>	0
File capacity	<i>numrec</i>	<i>filesize</i>	1023
Total number of extents	<i>numextents</i>	<i>numextents</i>	8
Extents initially allocated	<i>initalloc</i>	<i>initalloc</i>	1
:FILE command prohibition	(None)	Disallow :FILE Equation <i>foption</i> (Bits 5:1)	Allow :FILE command.
Dynamic file locking	(None)	Dynamic locking <i>aoption</i> (Bits 10:1)	Disallow dynamic locking.
Forms-alignment message	(None)	<i>formmsg</i>	No forms message sent.
User labels for disc file	(None)	<i>userlabels</i>	No user labels processed.

DEVICE-DEPENDENT CHARACTERISTICS

As noted previously throughout this section, for devicefiles, certain file characteristics are restricted by the devices on which the files reside. For instance, MPE always assigns a blockfactor of one to any file read from a card reader regardless of the blockfactor specified in your FOPEN call or :FILE command. For your convenience, all such device-dependent restrictions are summarized in table 6-12.

HEADERS AND TRAILERS. A facility for printing header and trailer records can be enabled by the Console Operator through the console command =HEADON. When this facility is enabled, and an output devicefile is directed to a card punch, MPE automatically punches a header card and a trailer card identifying the job that produced the file; if an output devicefile is directed to a line printer, MPE automatically prints header and trailer pages identifying the job that produced the file.

The Console Operator can disable the header facility by entering the =HEADOFF command.

Table 6-12. Device-Dependent Restrictions

INPUT ONLY DEVICES (SERIAL)

Card Reader/Paper Tape Reader

No carriage control (NOCCTL)

Undefined-length records

If card reader, ASCII only (can only read ASCII cards without using FCONTROL Intrinsic)

Blockfactor = 1

Domain = OLD permanent file in System Domain

If not ASCII, then NOBUF

If access mode = write-only, write-(save)-only, or append, then access violation results.

INPUT/OUTPUT DEVICES (PARALLEL)

Terminals

ASCII

NOBUF

Undefined-length records

Blockfactor = 1

INPUT/OUTPUT DEVICES (SERIAL)

Magnetic Tape Drive

No restriction other than no file label.

OUTPUT ONLY (SERIAL)

Line Printer/Card Punch/Paper Tape Punch/Plotter

Undefined-length records

Blockfactor = 1

Domain = NEW

Access Type = write-only (if read-only specified, access violation results)

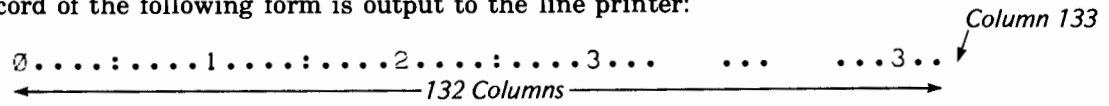
UNDEFINED (COMMON CHECKING)

If carriage control specified and not ASCII, access violation results

SPECIAL FORMS. When a program opens a new output devicefile, it may request special forms via the FOPEN intrinsic. This request transmits a user forms message to the operator's console, along with a request to mount the forms. The operator may respond as follows:

1. If the program specified a device class name for the file, the operator may allocate any unowned device in the class.
2. If the program specified a particular logical device number for the file, MPE asks the operator to mount the forms on the device requested if it is available.

When the operator allocates a line printer, MPE initiates a dialogue with him to align the forms. A standard record of the following form is output to the line printer:



This record is followed by a console message which asks the operator if the forms are properly aligned. This transaction is repeated until the operator indicates proper alignment. Now, the file can be output.

When a program closes a devicefile with special forms, MPE notifies the operator that the forms are no longer needed on the device.

If special forms are mounted on a device and a devicefile not requiring them is assigned to the device, MPE automatically asks the operator to mount standard forms or paper.

IMPLIED :FILE COMMANDS

Compilers and other subsystems that run under MPE accept actual file designators as parameters in the commands that call these programs. Whenever you furnish an actual file designator as a parameter for such a command, MPE implicitly issues a :FILE command that automatically equates the actual designator to a formal designator used within the subsystem. These implicit :FILE commands are discussed in more detail in Section VII.

NOTE

You can avoid using :FILE commands that conflict with certain :FILE commands pre-defined by the system if you know the formal file designators used by MPE subsystems and command executors. These designators are listed in Section VII, page 7-17.

CANCELLING :FILE COMMANDS

When you re-specify the programmatic or default specifications for a file by entering a :FILE command, that command normally remains in effect throughout the remainder of your session or job. Thus, if you enter a :FILE command equating the formal designator DATAFL to the actual designator CARDS (indicating a card file) and then run three programs that reference DATAFL, all three programs will access the file CARDS. You may, however, wish to nullify a previous :FILE command so that the referenced formal designator has the characteristics originally specified by the program using it. Although you may do this by issuing another :FILE command specifying the original characteristics, you can accomplish this even more conveniently through the :RESET command. For example, suppose you run two programs, both referencing a file called DFILE, a new temporary file on disc. Before you run the first program you want to re-define the file so that it is output to the standard list device. To do this, you issue a :FILE command equating DFILE with the actual designator \$STDLIST. In the second program, the file is again to be a temporary file on disc. You issue a :RESET command so that the specifications supplied by the second program (rather than those in the :FILE command) apply.

```
          :JOB JNAME,  UNAME.ANAME
          .
          .
          .
—→ :FILE DFILE=$STDLIST
      :RUN  PROG1
—→ :RESET DFILE
      :RUN  PROG2
          .
          .
          .
          :EOJ
```

NOTE

The :RESET command applies to files on disc, tape, or any other devices.

COPYING FILES

The MPE File Management System provides no commands for copying files into the system from devices such as card or paper-tape readers, for copying data from one disc file into another, or for writing the contents of a disc file onto a line printer or terminal. To do tasks such as these, you must use the *File Copier Subsystem (FCOPY)*, as directed in the reference manual covering that product. However, a method for dumping disc files to magnetic tape in a special format and later restoring them to the system is available under the File Management System to allow you to reliably back-up your data. This method is discussed under *Dumping Files Off-Line*, later in this section.

MANAGING DISC FILES

Although the MPE File Management System handles files that reside on any device, its true power and versatility lie in the use of files on disc. Disc files provide extensive storage capacity and rapid access in both sequential and direct (random) access mode. They can be made available to any user in the system or restricted by a flexible security system that screens accesses at the account, group, and file levels. The special operations available for disc files are described below. These include creating, saving, and deleting them; transferring them between groups, accounts, and other systems; and specifying security provisions for them.

CREATING A NEW FILE

When running a program that writes output to disc, you may need to ensure that an empty file is available for your data at the time the program begins. You can do this by entering the `:BUILD` command, which opens a new file on disc for you, immediately allocates space for the file, initializes this space by filling it with blanks (if it is an ASCII file) or zeros (if it is a binary file), and closes the file until your program opens it for use. Unless you explicitly specify otherwise, this is a permanent file. The `:BUILD` command is unlike the `:FILE` command used to override programmatic file specifications, in that the `:FILE` command does not take effect until your program opens the file; also, the `:FILE` command, by default, creates a temporary file.

NOTE

You can use the `:BUILD` command to create files only in your log-on account. Furthermore, if you are creating a permanent file, you must have `SAVE` access to the group to which the new file will belong; this access is assigned to you by your Account Manager.

Suppose you are running a program that needs a new session/job temporary file named `NFILE`, containing 500 records of 160 words each. The file is to have the default of eight extents, with one extent allocated immediately. You can provide this file by entering:

Number of records
↓
Record size
↓

```
:BUILD NFILE; DISC=500, REC=160; TEMP  Creates file  
:RUN DATASUB                               Runs program
```

You may want to structure your file so that all extents are restricted to a single disc device. This technique facilitates file access time, particularly where the file contains many extents. You can accomplish this by assigning the file to a disc identified by a class name to which no other device belongs, or by referencing the device by logical device number. For instance:

Logical device number
↓

```
:BUILD XFILE; REC=-128, 2, F, ASCII; DEV=12; DISC=5000, 15, 8  
:RUN PROGZ
```

LISTING FILE DESCRIPTIONS

If you want to find out what permanent disc files are available in the system file domain under your log-on group and account, simply enter the `:LISTF` command. The names of those files appear on your standard listing device as shown below:

```
:LISTF
```

```
FILENAME
```

```
DBPROG    DBTEST    FORMAINT  MYFILE    MYFILE1   MYFILE2 }  
MYNEWFL   MYPROG    MYPROGA  MYPROGX   SL         TESTERMP }  
TESTFORM  TSTEDITP  TSTFORMP
```

File names
↓

NOTE

The names of files in your session/job temporary file domain do not appear on this listing.

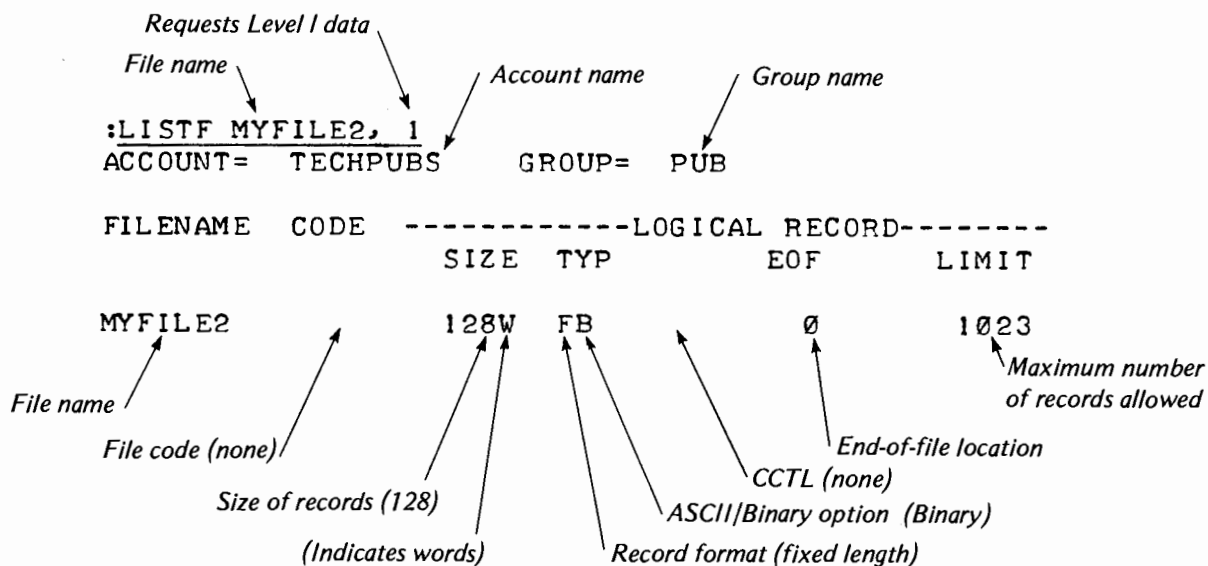
You can list the names of any or all files in the system, even though you may not have access to these files. For instance, if you want to find out what permanent files are available under the Public Group of the System Account, you name that group and account in your `:LISTF` command and specify the commercial-at sign (`@`) to indicate *all* files. (The `@` always denotes *all*, whether applied to files, groups, or accounts in this command.)

`:LISTF @.PUB.SYS`

FILENAME

ASEGMENT	ASEGPROC	BADTRACK	BASIC	BASICOMP	BILL
BUILDINT	BUILDSL	CMF	COBOL	COBOLB	CONFDATA
COPY	COST	COSTPROG	CPROG SYS	CROSSREF	CSDUMMY
CSHBSCØ	CSSBSCØ	DBDRIVER	DBDUMP	DBG SL	DELOAD
DBRESTOR	DBSCHEMA	DBSTORE	DBUNLOAD	DBUTIL	DECOMP
DEVDATA	DEVREC	DISKDUMP	DISKEDIT	DISPATCH	DPAN
DUMP3Ø	EDIT	EDITOLD	EDITOR	FCOPY	FINFO
FLEX	FORMAT	FORTRAN	FREE	FREE3Ø	GALLEY
GIVEFILE	IABA	ININ	INITIAL	IOCDPNØ	IOCDRDØ
IODSØ	IOFDISCØ	IOLPRTØ	IOMDISCØ	IOMDISC1	IOMDISK1
IOPLØTØ	IOPRPNØ	IOPTPNØ	IOPTRDØ	IOREMØ	IOSBSCØ
IOTAPEØ	IOTERMØ	JAN	JSYS1	JSYS2	JSYS3
LARRY	LARRYD	LISTCRET	LISTDIR	LISTDIR3	LISTEQ
LISTLOG	LISTLST	LOAD	LOADLIST	LOG	LOG2659
LOG266Ø	M78X	MARK			

For any file in the system, you can list certain additional details. For instance, suppose you wished to display the file code, record size and format, ASCII/binary and carriage-control options, current end-of-file location, and maximum number of records allowed in a file. You could do this as follows, indicating that you want to display *Level 1 Information*:



You can list an even greater level of detail by requesting the *Level 2 Information* option, as follows. This level includes blocking factor, number of disc sectors in use (including those used for file label and user header records), number of extents currently allocated, and maximum number of extents allowed.

```

      File name
      Requests Level 2 data
:LISTF MYFILE2, 2
ACCOUNT= TECHPUBS      GROUP= PUB

FILENAME  CODE  -----LOGICAL RECORD-----  SPACE-----  ACC
          SIZE  TYP          EOF          LIMIT R/B  SECTORS #X MX
MYFILE2           128W  FB          0          1023   1    128   1   8
                                     Blocking factor Sectors in use
                                     Extents allocated
                                     Maximum number of extents allowed

```

You can request that the file information be displayed on devices other than the standard listing device by naming the device desired in the `:LISTF` command. For instance, if you anticipate a lengthy listing and wish your terminal to remain free for other work while this listing is being produced, you might transmit the output to a line printer as follows:

```

:FILE PRTR; DEV=LP      Equates name PRTR with device class name LP.
:LISTF @.0,2;*PRTR     Directs Level 2 description of all files in all groups
                        of log-on account to PRTR. (Note back-reference
                        to :FILE command.)

```

The requested listing appears as follows:

```

ACCOUNT= TECHPUBS      GROUP= PUB

FILENAME  CODE  -----LOGICAL RECORD-----  SPACE-----  ACC
          SIZE  TYP          EOF          LIMIT R/B  SECTORS #X MX
DBPROG    PROG  128W  FB          6          1023   1    128   1
DBTEST    80B  FA          30          30   16    15   1
FORMAINT  PROG  128W  FB          64          64    1    65   1
MYFILE    128W  FB          0          1023   1    128   1
MYFILE1   128W  FB          0          1023   1    128   1
MYFILE2   128W  FB          0          1023   1    128   1

```

MPE obtains the information in the above displays from the file label. If you have System Manager Capability, you can list the entire label contents for any file in the system (*Level-1 Information*) written in octal code. If you have Account Manager Capability, you can request such a listing for all files in your account. To translate the octal contents of each file into ASCII code, you may run the program LISTDIR2, described in *MPE Utilities Reference Manual*. See also the `:LISTF` reference specifications and the *System Manager/System Supervisor Manual* for further information. The standard disc file label format is described in table 6-13.

Table 6-13. Disc File Label Contents

WORDS	CONTENTS
0-3	Local file name.
4-7	Group name.
8-11	Account name.
12-15	Identity of file creator.
16-19	File lockword.
20-21	File security matrix.
22 (Bits 0:15) (Bit 15:1)	Not used. File secure bit: If 1, file secured. If 0, file released.
23	File creation date.
24	Last access date.
25	Last modification date.
26	File code.
27	File control block vector.
28 (Bit 0:1) (Bit 1:1) (Bit 2:1) (Bit 3:1) (Bits 4:4) (Bits 8:6) (Bit 14:1) (Bit 15:1)	Store Bit. (If on, :STORE in progress.) Restore Bit. (If on, :RESTORE in progress.) Load Bit. (If on, program file is loaded.) Exclusive Bit. (If on, file is opened with exclusive access.) Device sub-type. Device type. File is open for write. File is open for read.
29 (Bits 0:8) (Bits 8:8)	Number of user labels written. Number of user labels.
30-31 32-34	Maximum number of logical records. Checksum.
35	Cold-load identity.
36	Foptions specifications.
37	Logical record size (in negative bytes).
38	Block size (in words).
39 (Bits 0:8) (Bits 1:5) (Bits 8:3)	Sector offset to data. Number of extents, minus 1. Disc flags.

Table 6-13. Disc File Label Contents (Continued)

WORDS	CONTENTS
40	Logical size of last block.
41	Extent size.
42-43	Number of logical records in file.
44-45	Address of first extent.
46-47	Address of second extent.
48-49	Address of third extent.
50-51	Address of fourth extent.
52-53	Address of fifth extent.
54-55	Address of sixth extent.
56-57	Address of seventh extent.
58-59	Address of eighth extent.
60-61	Address of ninth extent.
62-63	Address of tenth extent.
64-65	Address of eleventh extent.
66-67	Address of twelfth extent.
68-69	Address of thirteenth extent.
70-71	Address of fourteenth extent.
72-73	Address of fifteenth extent.
74-75	Address of sixteenth extent.
76-77	Address of seventeenth extent.
78-79	Address of eighteenth extent.
80-81	Address of nineteenth extent.
82-83	Address of twentieth extent.
84-85	Address of twenty-first extent.
86-87	Address of twenty-second extent.
88-89	Address of twenty-third extent.
90-91	Address of twenty-fourth extent.
92-93	Address of twenty-fifth extent.

Table 6-13. Disc File Label Contents (Continued)

WORDS	CONTENTS
94-95	Address of twenty-sixth extent.
96-97	Address of twenty-seventh extent.
98-99	Address of twenty-eighth extent.
100-101	Address of twenty-ninth extent.
102-103	Address of thirtieth extent.
104-105	Address of thirty-first extent.
106-107	Address of thirty-second extent.
125-128	Device class (in ASCII).

RENAMING A FILE

You can change the actual file designator of any permanent disc file in your log-on account if you are the original creator of that file. You may wish to do this, for instance, if you create a new file that you wish to assign an actual designator already in use for an existing file, and wish to save the existing file under a different designator. You could do this by using the `:RENAME` command to change the name of the old file, and then use the `:BUILD` command to create a new file with the old name:

New name assigned to existing file.
Old name assigned to existing file.
`:RENAME FILE1, FILEA`
`:BUILD FILE1`
Creates new disc file assigned name of old file.

ADDING, REMOVING, OR CHANGING LOCKWORDS

You can add or remove a lockword for an existing file by using the `:RENAME` command. For example, to add a lockword `MIXER` to the file named `ALPHA`, enter:

`:RENAME ALPHA, ALPHA/MIXER` ← *Lockword added*

To remove this lockword if it is already assigned, enter:

`:RENAME ALPHA/MIXER, ALPHA` ← *Lockword removed*

To alter a lockword, you can also use the :RENAME command, as follows:

Old lockword ↘ *New lockword* ↘
:RENAME ALPHA/MIXER, ALPHA/NEWMIX

NOTE

You may not use the :RENAME command for files in accounts other than your log-on account or for files that you have not created.

TRANSFERRING FILES

MPE provides facilities for transferring files between groups, accounts, and different systems.

INTER-GROUP TRANSFERS. To transfer a file from one group to another within the same account, use the :RENAME command, simply naming the new group in the second parameter. (You must be the original creator of the file to use this command.) For example:

Old group ↘ *New group* ↘
:RENAME MYFILE.GROUP1, MYFILE.GROUP2

NOTE

To use :RENAME in this way, you must have SAVE access to the group named in the second parameter (GROUP2 in the above example).

INTER-ACCOUNT TRANSFERS. To transfer a file from one account to another, proceed as follows:

1. Log-on to the computer under the account presently containing the file.
2. Enter the :RELEASE command to temporarily suspend any MPE security provisions covering the file. For instance:

:RELEASE FILEX ← *File name*

You can only enter this command if you are the original creator of the file.

3. Log-off from this account, and log-on under the account to which the file is to be transferred.
4. Run the File Copier Subsystem (FCOPY) to copy the file from the old account into this account. For instance,

New account name (optional entry) ↘
Old account name ↙

```
:RUN FCOPY.PUB.SYS  
>FROM=FILEX.GROUPA.ACCT1; NEW; TO=FILEX.GROUPA.ACCT2  
>EXIT
```

A copy of FILEX now exists under GROUPA of ACCT2; the original FILEX still exists under GROUPA of ACCT1. (ACCT2 must be your log-on account.)

5. Log-off from the present account and log-on again under the account containing the original file.
6. Restore the security provisions to the original file by entering the :SECURE command:

:SECURE FILEX

Or, if you want only one copy of the file in the system, delete the original file by entering the :PURGE command:

:PURGE FILEX

NOTE

To use the above commands, you must be the original creator of the file (unless you have Account Manager Capability).

Steps 1 through 3, and 5 through 6 can be avoided if the file security for ACCT1 (the old account) allows Read-Access from other accounts.

INTER-SYSTEM TRANSFERS. You can transfer one or more files between systems by copying them from their present system onto magnetic tape in a special format, transporting that tape to the new system, and loading the tape contents into the new system. To permit you to do this, however, the accounts, groups, and users to which the files belonged on the old system must also be defined on the new system. The technique for accomplishing this transfer involves the :STORE command (to write the files to tape) and the :RESTORE command (to copy the files from the tape into the new system.) This technique is also used to provide copies of files for system back-up, and is discussed under *Dumping Files Off-Line* and *Retrieving Dumped Files* later in this section.

You can also transfer files by copying them to tape via FCOPY and transporting that tape to the new system and loading it. The method for doing this is discussed in the reference manual covering FCOPY.

DELETING A FILE

You can remove from the system any permanent or temporary disc file to which you are allowed write access by the MPE file security system and the security defined by the person who created the file. To delete a permanent file, simply use the :PURGE command in the following manner:

:PURGE ALPHA	<i>Deletes file named ALPHA in log-on group.</i>
:PURGE MFILE.G30	<i>Deletes file named MFILE in group G30 of log-on account.</i>
:PURGE XFILE.G40.ACCT7	<i>Deletes file named XFILE in group G40 of account ACCT7.</i>

To delete a temporary file, include the TEMP parameter in the :PURGE command, as follows. If you omit this parameter, MPE searches for the file in the system file domain but not the session/job temporary file domain, and thus will fail to find the file, allowing it to remain in the system.

`:PURGE GAMMA, TEMP` ← *Denotes temporary file*

The :PURGE command operation is analagous to opening a file (FOPEN call) and then closing it (FCLOSE call) with delete disposition.

SAVING A FILE

Many MPE subsystems and other programs create temporary files that you may wish to use in subsequent sessions or jobs. To do this, you must save them as permanent files in the system file domain. If the subsystem or program that creates the file does not allow you the option to save it while the program is executing, you can save it by entering the :SAVE command. This method is most useful, for instance, if you are compiling or preparing programs onto the MPE default file for these operations, a temporary file that you reference under the name \$OLDPASS. As an example, if you use the :FORTPREP command to compile and prepare a program onto \$OLDPASS, you may wish to save this file so that you can run the compiled/prepared program in later sessions or jobs; if you do not do this, \$OLDPASS is destroyed at the end of the current session/job. To do this, you could use the :SAVE command as shown below. As part of this operation, you must specify a new file name for \$OLDPASS — MPE does not allow \$OLDPASS as a permanent file name. In the example below, the new file name XRUN is assigned. Now, no files named \$OLDPASS exist for the session/job.

```
:FORTPREP TEXTFL      Compiles and prepares program from source file TEXTFL.
:SAVE $OLDPASS, XRUN  Saves prepared program in file named XRUN.
```

NOTE

The file name \$OLDPASS always refers to the file most recently passed, whether created by a compiler, the MPE Segmenter, another subsystem, or a user program.

If you do not specify a group name as part of the new file designator, as in the above example, MPE saves the file under your log-on group. You can, however, specify any group to which you are allowed SAVE access as defined by the MPE file security system. This group, however, must always be one that is defined under your log-on account. For instance, to save the file \$OLDPASS under the new file name INTERIM in the Public Group (PUB) of your own account, enter:

```
:SAVE $OLDPASS, INTERIM.PUB
```

In addition to \$OLDPASS, MPE allows you to save any temporary file in your log-on account, under any group to which you have SAVE access. For instance, to save the temporary file DATAFILE in the group GROUPX, enter:

```
:SAVE DATAFILE.GROUPX
```


If the present name of the file is not \$OLDPASS, you cannot use the :SAVE command alone to change its name. Instead, you must also use the :RENAME command after the file is saved, as follows:

```
:SAVE DATAFILE.GROUPX           Saves file DATAFILE.  
:RENAME DATAFILE.GROUPX, DATABASE.GROUPX  Changes name to DATABASE.
```

NOTE

It is customarily good practice to save a file immediately after the program that creates the file is executed. Otherwise that file might be accidentally destroyed by another program or operation before it is made permanent.

The :SAVE operation is analagous to opening a file (FOPEN call) and then closing it (FCLOSE call) with permanent file disposition.

DUMPING FILES OFF-LINE

You can obtain a back-up copy of a particular user disc file or set of files by copying it off-line onto magnetic tape via the :STORE command. If you have standard user capability only, you can dump any file to which you have read-access. If a file has a negative file code, however, you must also have the Privileged Mode Optional Capability to dump this file. If you have the System Manager or System Supervisor Capability, you can dump any user file in the system. If you have Account Manager Capability, you can dump any file in your account (but cannot dump those with negative file codes unless you also have the Privileged Mode Capability). The files are copied in a special format along with all descriptive information (such as account name, group name and lockword), permitting them to be read back into the system later by the :RESTORE command.

The :STORE and :RESTORE commands are used primarily as a back-up for files. But you can also use them to interchange files between installations if the accounts, groups, and creators of the files to be restored are defined in the destination system. Furthermore, if you specify no destination device in the :RESTORE command, MPE does not guarantee which devices will actually receive the files — if a device of the same type as the original device with sufficient storage space cannot be found, the file is restored to any device that is a member of the device class DISC.

NOTE

Files with more than 16 disc extents cannot be restored under previous versions of MPE.

Files currently open for output, input/output, update, or append access cannot be acted upon by a :STORE command. Files currently being stored or restored cannot be acted upon by a :STORE command. However, files loaded into memory (containing currently running programs) and files open for input only can be stored, since their contents cannot be altered.

While a file is being dumped, it is *locked* by MPE so that it cannot be altered or deleted until safely copied to tape. If a session/job running a :STORE/:RESTORE function is aborted by yourself or the console operator, those files not yet stored or restored will be *unlocked* during the processing of the abort.

The flow chart in figure 6-8 shows the checks performed against a file to ensure its eligibility for dumping.

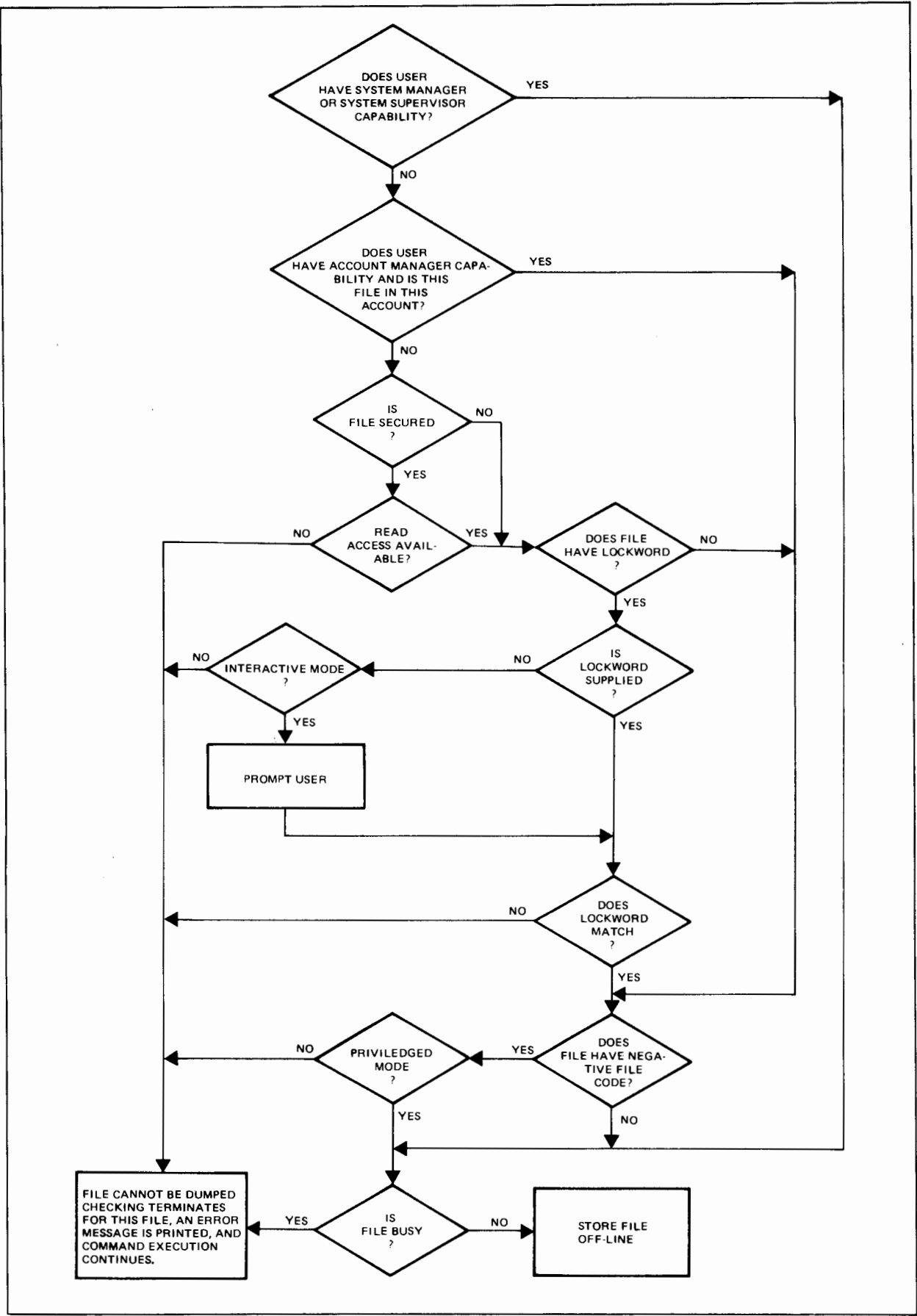


Figure 6-8. Checks for File Dump Eligibility

MAGNETIC TAPE FORMAT. Tapes produced by the :STORE command are compatible with those produced with the :SYSDUMP command, used by System Supervisors for general back-up of the overall system. (:SYSDUMP tapes are read by the MPE Initiator program to reload the system.) Tapes produced by either :STORE or :SYSDUMP are suitable as input to the :RESTORE command.

NOTE

In general, standard users use :STORE/:RESTORE when they desire to back-up only those files which belong to a particular set of groups or accounts. System Supervisors use :SYSDUMP for daily back-up of the overall system, since it provides a record of the latest accounting information. However, System Supervisors may also use :STORE/:RESTORE to save or load any or all files on the system provided the appropriate account, group, and user structures already exist.

The general format of a magnetic tape created by the :STORE command appears in figure 6-9. This format is defined more specifically in table 6-14. Both :STORE and :RESTORE support multi-file and multi-reel files.

The tape directory records are 12-word records, written at 85 records per block. The block size is 1020 words. (The last block of the tape directory may be shorter than 1020 words.) There is one entry (record) for each file on the tape. The entries are ordered the same as the files on the tape.

Each file on the tape is written in blocks 1024 words long. The last block may be shorter, but its length will be a multiple of 128 words. The first 128 words of the first block of each file contains the *file label* known to the file system.

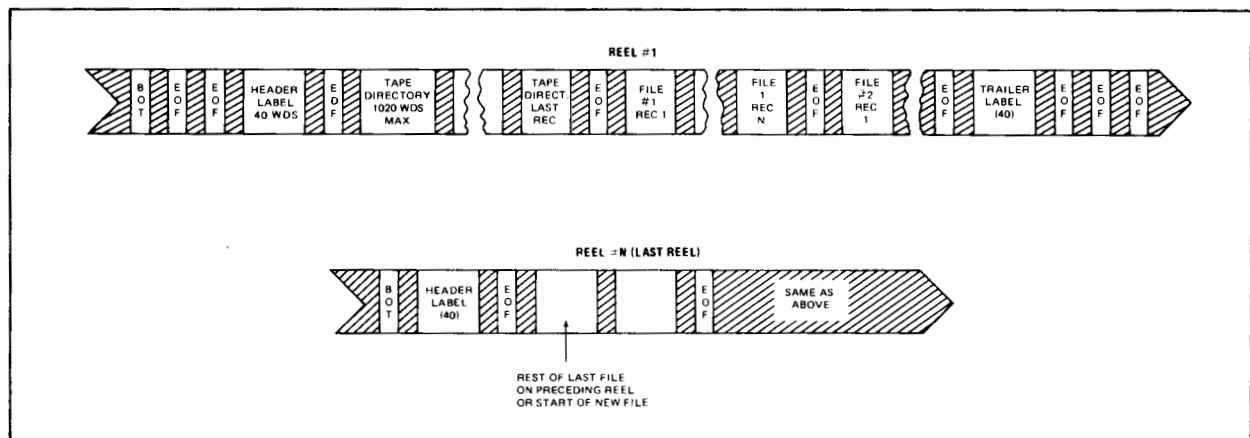


Figure 6-9. :STORE Tape Format

Table 6-14. :STORE Tape Format

ITEM NO.	ITEM																				
1	End-of-File (EOF) Mark																				
2	EOF Mark																				
3	Header label (40 words), used as follows: <table border="0" style="margin-left: 40px;"> <thead> <tr> <th style="text-align: left;">Words</th> <th style="text-align: left;">Contents</th> <th></th> </tr> </thead> <tbody> <tr> <td>0-13</td> <td>"STORE/RESTORE LABEL-HP/3000."</td> <td></td> </tr> <tr> <td>14-22</td> <td>Unused by MPE.</td> <td></td> </tr> <tr> <td>23</td> <td>Reel Number.</td> <td></td> </tr> <tr> <td>24</td> <td>Bits (0:7) = last 2 digits of year (7:9) = Julian date</td> <td rowspan="2">} Date of creation.</td> </tr> <tr> <td>25</td> <td>Bits (0:8) = hours (8:8) = minutes</td> </tr> <tr> <td>26</td> <td>Bits (0:8) = seconds (8:8) = tenth-of-seconds</td> <td>} Time of creation.</td> </tr> </tbody> </table>	Words	Contents		0-13	"STORE/RESTORE LABEL-HP/3000."		14-22	Unused by MPE.		23	Reel Number.		24	Bits (0:7) = last 2 digits of year (7:9) = Julian date	} Date of creation.	25	Bits (0:8) = hours (8:8) = minutes	26	Bits (0:8) = seconds (8:8) = tenth-of-seconds	} Time of creation.
Words	Contents																				
0-13	"STORE/RESTORE LABEL-HP/3000."																				
14-22	Unused by MPE.																				
23	Reel Number.																				
24	Bits (0:7) = last 2 digits of year (7:9) = Julian date	} Date of creation.																			
25	Bits (0:8) = hours (8:8) = minutes																				
26	Bits (0:8) = seconds (8:8) = tenth-of-seconds	} Time of creation.																			
4	EOF Mark																				
5	Tape directory—Consists of 12-word entries (records), blocked 85 per 1020—word block. (The last block may be shorter.) There is one entry for each file on the tape, and the entries are ordered the same as the files. The 12-word entry is: <table border="0" style="margin-left: 40px;"> <thead> <tr> <th style="text-align: left;">Word</th> <th style="text-align: left;">Contents</th> </tr> </thead> <tbody> <tr> <td>0-3</td> <td>File name</td> </tr> <tr> <td>4-7</td> <td>Group name</td> </tr> <tr> <td>8-11</td> <td>Account name</td> </tr> </tbody> </table>	Word	Contents	0-3	File name	4-7	Group name	8-11	Account name												
Word	Contents																				
0-3	File name																				
4-7	Group name																				
8-11	Account name																				
6	EOF Mark																				
7	First file. The data is blocked with 1024 words per block. (The last block may be shorter, but will always be a multiple of 128 words.) For fixed-length and undefined-length record files, only data up to the end-of-file is dumped; intervening zero-length extents are not dumped. For variable-length record files, only allocated extents are dumped.																				
8	EOF Mark																				
9	Second File																				
10	EOF Mark . . .																				
11	Last File																				
12	EOF Mark																				
13	Trailer Label (40 words). Identical to header label (Item 3) except that Words 21 and 22 are used as follows:																				



Table 6-14. :STORE Tape Format (Continued)

ITEM NO.	ITEM	
	Word	Use
	21	= 1 means that preceding file ended with preceding EOF mark
	22	= 1 means that entire tape set ends with preceding EOF mark
14	EOF Mark	
15	EOF Mark	
16	EOF Mark	

:STORE tapes may have multiple reels. If end-of tape (EOT) is detected during a write data operation, a file mark is written followed by Items 13 to 16 above, with word 21 of the trailer label set to 1 if this was the last record of the file and 0 otherwise. If EOT is detected on a write file mark operation, Items 13 to 16 are written, with word 21 set to 1 and word 22 set to 1 if this is the last file on the tape, and 0 otherwise. Reels subsequent to Reel 1 have the following format:

1. Header label
2. EOF mark
3. Remainder of preceding file or next file
4. EOF mark
5. Next file; the rest of the tape is written in the same format as the first reel.

LISTING OUTPUT. After the tape is written, MPE prints data showing the results of the :STORE command. By default, this output is sent to the standard list device (\$STDLIST). However, you can override this default and transmit the output to another file by issuing a :FILE command equating SYSLIST (the formal designator by which the :STORE command executor references this list file) to another file. For example, if you are located at a terminal, you might transmit this output to a line printer by entering:

```
:FILE SYSLIST=MYFILE; DEV=LP
```

(Assume the device class LP indicates a high-speed line printer.)

If you omit the SHOW parameter from the :STORE command, only the total number of files actually stored, a list of files not stored, and a count of files not stored, are printed. But if SHOW is included, the listing of files appears in the format shown in figure 6-10. A sample print-out is shown in the lower portion of the figure. In this format, *xxx* is a value denoting the total number of files dumped onto tape; *yyy* denotes the number of files requested that were not dumped. The notations *filename*, *groupname*, and *acctname* under the *FILES STORED* heading name the individual files dumped, and their groups and accounts, respectively. The notation *ldn* indicates the logical device number (in decimal) of the device on which the file label and first extent resides, and *addr* is the absolute address (in octal) of the file label. The notations *filename*, *groupname*, and *acctname* under the *FILES NOT STORED* heading, indicate the individual files not dumped, and their groups and accounts. The notation *fileset#* shows the number of the fileset to which the particular file belongs (relative to its position following the :STORE command name). The notation *msg* is a message denoting the reason that the file was not dumped. (These errors do not abort the file storing operation, which continues.) Table 6-15 lists the messages and their meanings.

FILES STORED = xxx

<i>FILE</i>	<i>.GROUP</i>	<i>.ACCOUNT</i>	<i>LDN</i>	<i>ADDRESS</i>
<i>filename1</i>	<i>.groupname1</i>	<i>.acctname1</i>	<i>ldn1</i>	<i>addr1</i>
<i>filename2</i>	<i>.groupname2</i>	<i>.acctname2</i>	<i>ldn2</i>	<i>addr2</i>
<i>.</i>				
<i>filenamen</i>	<i>.groupnamen</i>	<i>.acctnamen</i>	<i>ldnn</i>	<i>adrn</i>

FILES NOT STORED = yyy

<i>FILE</i>	<i>.GROUP</i>	<i>.ACCOUNT</i>	<i>FILESET</i>	<i>REASON</i>
<i>filename1</i>	<i>.groupname1</i>	<i>.acctname1</i>	<i>fileset</i>	<i>msg</i>
<i>filename2</i>	<i>.groupname2</i>	<i>.acctname2</i>	<i>fileset</i>	<i>msg</i>
<i>.</i>				
<i>filenamen</i>	<i>.groupnamen</i>	<i>.acctnamen</i>	<i>fileset</i>	<i>msg</i>

Example

FILES STORED = 13

FILE	.GROUP	.ACCOUNT	LDN	ADDRESS
DATA	.PUB	.SUPPORT	1	%24324
FSMT	.PUB	.SUPPORT	1	%111052
FSMTS	.PUB	.SUPPORT	1	%110775
FTEST	.PUB	.SUPPORT	1	%111237
FTESTJ1	.PUB	.SUPPORT	1	%24560
FTESTJ2	.PUB	.SUPPORT	1	%24642
FTESTJ3	.PUB	.SUPPORT	1	%25542
FTESTJOB	.PUB	.SUPPORT	1	%23463
FTESTJX	.PUB	.SUPPORT	1	%41207
FTESTS	.PUB	.SUPPORT	1	%23603
JUNKJOB	.PUB	.SUPPORT	1	%23533
PEOF	.PUB	.SUPPORT	1	%374040
PEOFS	.PUB	.SUPPORT	1	%373737

FILES NOT STORED = 1

FILE	.GROUP	.ACCOUNT	FILESET	REASON
K2861445	.PUB	.SUPPORT	1	BUSY

Figure 6-10. List Output Format of :STORE Command

Table 6-15. :STORE Command Error Messages

MESSAGE	MEANING
ACCOUNT NOT IN DIRECTORY	Specified account does not exist.
GROUP NOT IN DIRECTORY	Specified group does not exist.
FILE NOT IN DIRECTORY	Specified file does not exist.
BUSY	File is open for output, or is currently being stored or restored.
FILE CODE < 0 AND NO PRIVMODE	You tried to store a file with a negative file code, but do not have Privileged Mode Capability.
LOCKWORD WRONG	The file lockword either was not provided or was specified incorrectly.
READ ACCESS FAILURE	You do not have read access to the specified file.
FILE LABEL ERROR	Due to a problem beyond your control, the file label is not valid.
CATASTROPHIC ERROR	One of the following errors occurred, and the :STORE command is aborted: command syntax error; disc input/output error (in system); file directory error; file system error on tape file (named TAPE), list file (named LIST), or temporary disc files (named GOOD and ERROR) used by :STORE command executor.

EXAMPLES. The following example illustrates how to make a back-up copy of files. To copy all files in the group GP4M (in your log-on account) to a tape file named BACKUP, enter the following commands. A listing of files copied and not copied appears on the standard listing device.

```
:FILE BACKUP; DEV=TAPE
:STORE @.GP4M; *BACKUP; SHOW
```

Explicit or implicit redundant references are permitted among the files you request, but once a file has been locked down for dumping, any subsequent reference to it results in the message BUSY even though execution of the :STORE command continues. For instance, suppose the file identified as FN.GN.AN is a member of the fileset referenced by @ in the following command:

```
:STORE @, FN.GN.AM; *DUMPTAPE; SHOW
```

The command is executed successfully, but the *fileset#* and *msg* notations under *FILES NOT STORED* on the listing show:

<u>filename</u>	<u>groupname</u>	<u>acctname</u>	<u>fileset#</u>	<u>msg</u>
FN	GN	AN	2	BUSY

This same file is also noted under FILES STORED. (The file is, in fact, actually stored.)

RETRIEVING DUMPED FILES

You can copy back into the system, onto disc, a particular file or fileset that has been stored off-line on tape by :STORE (or :SYSDUMP). If you have System Manager or System Supervisor Capability, you can restore any file from a :STORE tape, assuming the account and group to which the file belongs, and the user who created the file exist in the system. If you have Account Manager Capability, you can restore any file in your account (but cannot restore those with negative file codes unless you also have the Privileged Mode Capability). If you have standard user capability, you can restore any tape file in your log-on account if you have SAVE access to the group to which the file belongs (but you cannot restore those with negative file codes unless you also have Privileged Mode Capability). If the file on tape is protected by a lockword, you must supply the lockword in the :RESTORE command. (If you are logged-on interactively, you are prompted for the lockword if you fail to supply it.)

MPE attempts to restore the files to a device of the same class as the device on which they were originally created. The files are attached to the appropriate groups and accounts with previous account and group names and lockwords all reinstated. *The :RESTORE command does not create any new accounts or groups. Any tape file to be restored is restored only if the account name and group name exist on disc (in the system directory).*

If a copy of a file to be restored already exists on disc, you must have write access to the disc file (since it will be purged by :RESTORE) unless you use the KEEP keyword. This keyword specifies that if a file referenced in the :RESTORE command currently exists on disc, the file on disc is retained and the corresponding tape file is not copied into the system.

Files currently open, loaded into memory, or being stored or restored, cannot be acted upon by a :RESTORE command.

The :RESTORE command performs the same checking done by the :STORE command, to ensure a file's eligibility for retrieval. If you include the SHOW parameter in the :RESTORE command, MPE prints a listing showing which files were restored. Otherwise, a count of files restored, a list of files not restored, and a count of files not restored, are supplied.

LISTING OUTPUT. As with the listing produced by :STORE, the listing output by :RESTORE is transmitted to a file whose formal designator is SYSLIST; if you do not specify otherwise, this file is equated, by default, to the standard list device (\$STDLIST). An example of a typical :RESTORE operation with SHOW and KEEP options appears in figure 6-11. The format of the listing is the same as that for the :STORE example shown in figure 6-10. The notation *msg* is an error message denoting the reason that the file was not restored. (These errors do not abort the file-restoring operation.) The messages and their meanings are listed in table 6-16.

FILES RESTORED = 6

FILE	.GROUP	.ACCOUNT	LDN	ADDRESS
DATA	.PUB	.SUPPORT	1	%23463
FTEST	.PUB	.SUPPORT	1	%24324
FTESTJ1	.PUB	.SUPPORT	1	%23741
FTESTJOB	.PUB	.SUPPORT	1	%23753
FTESTS	.PUB	.SUPPORT	1	%23765
PEOF	.PUB	.SUPPORT	1	%24354

FILES NOT RESTORED = 7

FILE	.GROUP	.ACCOUNT	FILESET	REASON
FSMT	.PUB	.SUPPORT	1	ALREADY EXISTS
FSMTS	.PUB	.SUPPORT	1	ALREADY EXISTS
FTESTJ2	.PUB	.SUPPORT	1	ALREADY EXISTS
FTESTJ3	.PUB	.SUPPORT	1	ALREADY EXISTS
FTESTJX	.PUB	.SUPPORT	1	ALREADY EXISTS
JUNKJOB	.PUB	.SUPPORT	1	ALREADY EXISTS
PEOFS	.PUB	.SUPPORT	1	ALREADY EXISTS

Figure 6-11. List Output of :RESTORE with SHOW and KEEP

Table 6-16. :RESTORE Command Error Messages

MESSAGE	MEANING
ACCOUNT DIFFERENT FROM LOGON	The file's account name is different from the name of your log-on account. (Users do not have save-access to groups outside of their log-on accounts.)
ACCOUNT DISC SPACE EXCEEDED	The account's disc space limit would be exceeded by restoring this file.
ACCOUNT NOT IN DIRECTORY	The account specified does not exist in the system.
ALREADY EXISTS	A copy of the file specified already exists on disc, and KEEP was also specified. The file was not replaced.
BUSY	The disc file is open, loaded, or being stored or restored at present.
CATASTROPHIC ERROR	A catastrophic error occurred while the system was restoring either this file or one previous to it on the tape, and the :RESTORE command was aborted. This message may result from one of the following: Command Syntax error. Disc input/output error (in system).

Table 6-16. :RESTORE Command Error Messages (Continued)

MESSAGE	MEANING
	<p>File directory error.</p> <p>File system error on the tape file (TAPE), list file (LIST), or any of the three temporary files (GOOD, ERROR, and CANDIDAT) used by the :RESTORE command executor.</p> <p>Improper tape; the tape used for input was not written in :STORE/:RESTORE format.</p> <p>No continuation reel; the computer operator could not find a continuation reel for a multi-reel tape set.</p> <p>Device reference error; the specification for the <i>device</i> parameter is illegal, or the device requested is not available.</p> <p>Tape read error in a sensitive part of the tape, which makes it impossible to continue. (Most tape errors are merely skipped, omitting the affected file.)</p>
CREATOR NOT IN DIRECTORY	The creator of the file is not defined in the system.
DISC FILE CODE < 0 and NO PRIV MODE	One of the files (on disc) to be replaced has a negative file code, and you do not have Privileged Mode Capability.
DISC FILE LOCKWORD WRONG	The disc file has a lockword that does not match the lockword for the file on tape.
FILE LABEL ERROR	Due to a problem beyond your control, the file label is not valid.
GROUP DISC SPACE EXCEEDED	The group's disc space limit would be exceeded by restoring this file.
GROUP NOT IN DIRECTORY	The group specified does not exist in the system.
NOT ON TAPE	The file specified is not on the tape.
OUT OF DISC SPACE	There is insufficient disc space to restore this file.
SAVE ACCESS FAILURE	You do not have save-access to the group to which the file belongs.
TAPE FILE CODE < 0 and NO PRIV MODE	One of the files (on tape) to be restored has a negative file code, and you do not have Privileged Mode Capability.
TAPE FILE LOCKWORD WRONG	The tape file has a lockword that you did not supply or did not specify correctly.
TAPE READ ERROR	A tape read error has occurred on a block other than that containing the file label.
WRITE ACCESS FAILURE	You do not have write-access to the copy of the file on disc.

NOTE

Tapes created by the `:STORE` command and the `:SYSDUMP` command are compatible. (`:SYSDUMP` is discussed in the *System Manager/System Supervisor Manual*.) Thus, a tape dumped via `:SYSDUMP` can be used as input for the `:RESTORE` command. However, `:STORE/:RESTORE` tapes cannot be used as the first reel of system-initiation tapes during a reload, since the operating system has not been copied to the tape.

EXAMPLE. The following example shows how to retrieve files using `:RESTORE`. To retrieve from the file named `BACKUP` all files formerly belonging to your log-on group, enter:

```
:FILE BACKUP; DEV=TAPE
:RESTORE *BACKUP; @; KEEP; DEV=MDISC; SHOW
```

If a file satisfying the `@` specification already exists in the system, it is not restored.

SPECIFYING DISC FILE SECURITY

As previously noted, when you log onto the system or submit a batch job to it, you are related to an account and to a group of files owned by that account. Associated with each account, group, and individual file, there is a set of security provisions that specifies any restrictions on access to the files in that account or group, or to that particular file. (Notice that these provisions apply to disc files only.) These restrictions are based upon two factors:

1. Modes of Access (reading, writing, or saving, for example).
2. Types of Users (users with Account Librarian or Group Librarian Capability, or creating users, for instance) to whom the access modes specified are permitted.

The security provisions for any file describe *what modes of access* are permitted to *which users* of that file.

The access modes possible, the mnemonic codes used to reference them in MPE commands relating to file security, and the complete meanings of these modes, are listed below:

Access Mode	Mnemonic Code	Meaning
Reading	R	Allows users to read files.
Locking	L	Permits a user to prevent concurrent access to a file by himself and another user, allowing accessing user to <i>lock</i> file while he is using it. Specifically, it permits use of the <code>FLOCK</code> and <code>FUNLOCK</code> intrinsics, and the exclusive-access option of the <code>FOPEN</code> intrinsic, all described in <i>MPE Intrinsics Reference Manual</i> .

Appending	A	Allows users to add information and disc extents to files, but prohibits them from altering or deleting information already written. This access mode implicitly allows the locking (L) access mode-described above.
Writing	W	Allows users general writing access, permitting them to add to, delete, or change any information on files. This includes removing entire files from the system with the :PURGE command. Writing access also implicitly allows the locking (L) and appending (A) access modes described above.
Saving	S	Allows users to declare files <i>within their groups</i> permanent (via the :SAVE command or FCLOSE intrinsic), and to rename such files (via the :RENAME command). This ability includes the creation of a new permanent file with the :BUILD command.
Executing	X	Allows users to run programs stored on files, with the :RUN command or CREATE intrinsic (described in <i>MPE Intrinsic Reference Manual</i>).

The types of users recognized by the MPE security system, the mnemonic codes used to reference them, and their complete definitions, are listed below.

User Type	Mnemonic Code	Meaning
Any User	ANY	Any user defined in the system; this includes all categories defined below.
Account Librarian User	AL	User with Account Librarian Capability, who can manage certain files within his account that may or may not all belong to one group.
Group Librarian	GL	User with Group Librarian Capability, who can manage certain files within his home group.
Creating User	CR	The user who created this file.
Group User	GU	Any User allowed to access this group as his log-on or home group, including all GL users applicable to this group.
Account Member	AC	Any user authorized access to the system under this account; this includes all AL, GU, GL, and CR users under this account.

Users with System or Account Manager Capability bypass the standard security mechanism; a System Manager always has unlimited (R,A,W,L,X) access to any file in the system, and S access to any group in his log-on account; an Account Manager always has unlimited (R,A,W,L,X) access to any file in his log-on account, and S access to any group in his log-on account.

The user-type categories that a user satisfies depend on the file he is trying to access. For example, a user accessing a file that is not in his home group is not considered a group librarian for this access even if he has the Group Librarian user attribute.

Notice that in order to extend a file, either W and A access to that file is required.

NOTE

Remember that, in addition to the above restrictions, in force at the account, group, and file level, a file lockword can be specified for any file. Users must then specify the lockword along with the filename to access this file. The way in which lockwords are assigned to files is discussed earlier in this section.

The security provisions for the account and group levels are managed only by users with the System Manager and the Account Manager Capabilities respectively, and can only be changed by those individuals. The manner in which they are implemented is described in *System Manager/System Supervisor Manual*. Because they also relate to the security provisions at the file level (which are the responsibility of the standard user), the account and group level provisions are also summarized in this section.

ACCOUNT-LEVEL SECURITY

The security provisions that broadly apply to all files within an account are set by a user with System Manager Capability when he creates the account. The initial provisions can be changed at any time, but only by that user.

At the account level, five access modes are recognized:

Reading (R)
Appending (A)
Writing (W)
Locking (L)
Executing (X)

Also, at the account level, two user types are recognized:

Any User (ANY)
Account Member (AC)

If no security provisions are explicitly specified for the account, the following provisions are assigned by default:

- For the system account (named SYS), though which the System Manager initially accesses the system, reading and executing access are permitted to all users; appending, writing, and locking access are limited to account members. Symbolically, these provisions are expressed as follows:

R,X:ANY; A,W,L:AC

In this format, colons are interpreted to mean “. . . is permitted only to. . .”, or “. . . is limited to. . .”. Commas are used to separate access modes or user types from each other. Semicolons are used to separate entire access mode/user type groups from each other.

- For all other accounts, the reading, appending, writing, locking, and executing access are limited to account members. (R,A,W,L,X: AC).

GROUP-LEVEL SECURITY

The security provisions that apply to all files within a group are initially set by an Account Manager when he creates the group. They can be equal to or more restrictive than the provisions specified at the account level. (The group’s security provisions also can be less restrictive than those of the account — but this effectively results in *equating* the group restrictions with the account restrictions, since a user failing security checking at the account level is denied access at that point, and is not checked at the group level.) The initial group provisions can be changed at any time, but only by an Account Manager for that group’s account.

At the group level, six access modes are recognized:

Reading (R)
Appending (A)
Writing (W)
Locking (L)
Executing (X)
Saving (S)

Also, at the group level, five user types are recognized:

Any User (ANY)
Account Librarian User (AL)
Group Librarian User (GL)
Group User (GU)
Account Member (AC)

If no security provisions are explicitly specified, the following provisions apply by default.

- For a public group (named PUB), whose files are normally accessible in some way to all users within the account, reading and executing access are permitted to all users; appending, writing, saving, and locking access are limited to Account Librarian Users and Group Users (including Group Librarian Users). (R,X:ANY;A,W,L,S:AL,GU).
- For all other groups in the account, reading, appending, writing, saving, locking, and executing access are limited to group users. (R,A,W,L,X,S:GU).

FILE-LEVEL SECURITY

When a file is created, the security provisions that apply to it are the default provisions assigned by MPE at the file level, coupled with the user-specified or default provisions assigned to the account and group to which the file belongs. At any time, however, the creator of the file (and *only* this user) can change the file-level security provisions. Thus, the total security provisions for any file depend upon specifications made at all three levels — the account, group, and file levels. To successfully access a file in the requested mode, you must pass tests at all three levels — account, group, and file security, in that order.

If no security provisions are explicitly specified for a file by its creator, MPE assigns the following provisions at the file level by default:

- For all files, reading, appending, writing, locking, and executing access are permitted to all users. (R,A,W,L,X:ANY).

Because the total security for a file always depends on security at all three levels, a file not explicitly protected from a certain access mode at the file level may benefit from the default protection at the group level. For example, the default provisions at the file level allow the file to be read by any user — but the default provisions at the group level allow access only to group users. Thus, the file can only be read by a group user.

In summary, the default security provisions at the account, group, and file levels combine to result in these overall default security provisions:

Filereference	File	Access Permitted	Save Access to Group
filename.PUB.SYS	Any file in Public Group of System Account	(R,X:ANY; W:AL,GU)	AL,GU
filename.group-name.SYS	Any file in any group in System Account.	(R,W,X:GU)	GU
filename.PUB.ac-countname	Any file in Public Group of any account.	(R,X:AC; W:AL,GU)	AL,GU
filename.group-name.accountname	Any file in any group in any account.	(R,W,X:GU)	GU

Stated another way, when the default security provisions are in force at all levels and you are a standard user (without any other user attributes), you have:

- Unlimited access (in all modes) to all files in your log-on group and home group.
- Reading and executing access (only) to all files in the public group of your account and the public group of the System Account.

You cannot access any other file in the system (in any mode).

You can only create files within your own account.

MPE determines the legitimacy of a request to access a file by checking the access mode you request against the final access mode authorized for users of your type by the file security provisions. For commonly-used intrinsics, the functions requested versus the access-mode necessary for the functions to be honored are shown in table 6-17. (The intrinsics themselves are described in detail in *MPE Intrinsic Reference Manual*.)

Table 6-17. Access Modes for Intrinsics

INTRINSIC	FUNCTION REQUESTED	ACCESS MODE REQUIRED TO HONOR FUNCTION
FOPEN	Reading Writing Appending (only) Input/output Updating Exclusive accessing Semi-exclusive accessing	R W A (or W) R and W R and W L (or W or A) L (or W or A)
FLOCK	File locking	L (or W or A)
FUNLOCK	File unlocking	L (or W or A)
FCLOSE	Saving Any access beyond the current end-of-file.	S (relative to desired group) A or W

Additionally, any attempt to access a file beyond the current end-of-file indicator requires permission for A or W access mode.

When a file is closed with permanent file disposition by the FCLOSE intrinsic with a *seccode* (security code) parameter entry of 0 for normal MPE security, the security provisions assigned are:

R,A,W,L,X: ANY

But when this is done with a *seccode* parameter of 1 for private user file security, the security provisions assigned are:

R,A,W,L,X:CR

Then, if the user desires, he can assign more or less restrictive security provisions with the :ALTSEC command, described below.

CHANGING SECURITY PROVISIONS

You can change the security provisions for any disc file that you have created. You do this by using the :ALTSEC command, which permanently deletes all previous provisions specified for this file at the file

level, and replaces them with those defined as the command parameters. This command does not, however, affect any account-level or group-level provisions that may cover the file. Furthermore, it does not affect the security provided by the *lockword* (if one exists).

For example, suppose you want to alter the security provisions for the file FILEX to permit the ability to read, execute, and append information to the file only to the creating user and the log-on or home-group users. You can do this with the following :ALTSEC command. (Notice that in the *modelist:userlist* parameter group, the separating colon can be interpreted as indicating “. . . is permitted only to . . .”. Thus, the parameter group in this command implies “The Appending, Reading, and Executing Modes are permitted only to the Creating User and Log-On and Home-Group Users.”)

```
:ALTSEC FILEX; (A,R,X: CR, GU)
```

To restore the default security provisions to this file, you would enter:

```
:ALTSEC FILEX
```

Suppose you have created a file named FILEZ for which you have allowed yourself program-execute access only. You now wish to change this file's security provisions so that any group user can execute the program stored within it, but only the group librarian can read and write on it. Even though you do not have read or write access to the file, you can still alter its security provisions by entering:

```
:ALTSEC FILEZ; (X:GU; R,W:GL)
```

You always retain the ability to change the security provisions of a file that you have created, even when you are not allowed to access the file in any mode. Thus, you can even change the provisions to allow yourself access.

SUSPENDING AND RESTORING SECURITY PROVISIONS

From time to time, you may wish to suspend all account, group, and file-level security provisions governing a disc file that you have created. This allows the file to be accessed in any mode by any user; in other words, it offers unlimited access to the file. (Note that this is a temporary suspension that does *not* require you to have System or Account Manager Capability.) You suspend the security provisions by entering the :RELEASE command. (File lockword protection, however, is not removed by this command.) The :RELEASE command does not modify the file security settings recorded in the system; it merely bypasses them temporarily. (The :RELEASE command remains in effect until you enter the :SECURE command, described below, in this or a later session/job.)

To release the security provisions for the file named FILEZ in your log-on group, enter:

```
:RELEASE FILEZ
```

If the file has a lockword and you wish to remove that as well as all account, group, and file level security provisions, you must use the :RENAME command as well as the :RELEASE command.

```
:RENAME FILEZ/LOCKZ, FILEZ    Removes lockword.  
:RELEASE FILEZ                Removes security provisions.
```

To restore the security provisions to their previous settings, and the lockword as well, enter:

```
:SECURE FILEZ                               Restores security provisions.  
:RENAME FILEZ, FILEZ/LOCKZ                 Restores lockword.
```

READING DATA FROM OUTSIDE STANDARD INPUT STREAM

Occasionally, sessions and jobs require input data that must be entered via devicefiles other than your standard session/job input file. This occurs, for instance, when you wish to read input from a batch input device while running an interactive session. You can obtain this data in either of two ways:

1. By entering the data from a file that begins with the :DATA command, followed by the data, and terminating with the :EOD command, as directed below. (Data entered in this way can only be submitted via a device configured to accept the :DATA command.) If done properly, this method requires no dialogue between yourself and the Console Operator.
2. By simply requesting the data in your program, which transmits a message to the Console Operator asking him to load the data into an appropriate device as soon as one is available, and allocate that device. (Data submitted in this way begins with your first data record and can only be entered through a device *not* configured to accept the :DATA, :HELLO, or :JOB commands, or one for which the Console Operator has entered the = REFUSE command.)

As an example, suppose you are running a session identified by the session name SESSA, user name BROWN, and account name ACCT1. You wish to make data on punched cards available to that session, to be used by a program named PROGY. This program references the data under the formal file designator DATAFL. Proceed as follows:

1. Arrange your data deck beginning with the :DATA command and terminating with the :EOD command, as follows:

```
:DATA SESSA,BROWN.ACCT1  
.  
.  
.  
(YOUR DATA)  
.  
.  
.  
:EOD
```

2. Load the cards into the card reader and make the reader ready. If the card reader is a spooled device, the data is copied to disc, where it awaits your access. If it is not a spooled device, the :DATA command is read but the subsequent data remains in the card reader until your program accesses it. The :DATA command is used to build an entry in the device directory that identifies the file to the system.
3. Begin your session, making sure that you use *precisely* the same session identity entered in your :DATA command. (For instance, if you omit the optional session name SESSA, your session will not be able to access the data.) To log-on, enter:

```
:HELLO SESSA,BROWN.ACCT1
```

Same identity used in :DATA command.

4. ENTER a :FILE command equating the formal file designator DATAFL with the card reader. For instance,

```
:FILE DATAFL; DEV=CARD
```

5. Run the program that requires the data; when the program attempts to read the data, it will be available:

```
:RUN PROGY
```

NOTE

If you have not entered the data through the card reader prior to this step, the program transmits a message to the Console Operator requesting this data and waits for him to furnish it. The operator may, at his option, supply the data via the :DATA command or by allocating a device configured not to accept this command; or if he does not have the data, he may send you a message asking you to supply it or he may abort the session.

6. Once the data has been read, it is no longer available to the system. When you run another program requiring this data, you must therefore enter the data again in the above manner.

MPE permits you to submit data to a single session/job via two or more separate devicefiles, each identified by a unique file name. To do this, you simply furnish the file name as part of each :DATA command. For example, suppose the above session requires two files, each from a different card reader. You would proceed as follows:

1. Arrange each data deck beginning as follows, indicating the unique file name in each :DATA command.

```
File name
  |
  v
:DATA SESSA,BROWN.ACCT1; DATAFL1
  .
  .
  .
  (YOUR DATA)
  .
  .
  .
:EOD
```

} First data deck

```
File name
  |
  v
:DATA SESSA,BROWN.ACCT1; DATAFL2
  .
  .
  .
  (YOUR DATA)
  .
  .
  .
:EOD
```

} Second data deck

2. Place the cards in the separate card readers, and make the readers ready (or submit the cards to a spooled card reader).
3. Begin your session as follows:

```
:HELLO SESSA, BROWN.ACCT1
.
.
.
```



4. Enter :FILE commands to identify the card readers, as follows:

```
:FILE DATAFL1; DEV=CARD
:FILE DATAFL2; DEV=CARD
```

Notice that you do not need to supply the logical device numbers of the card readers to uniquely identify them — the file names are sufficient.

5. Run the program that uses the data:

```
:RUN PROGY
```

NOTE

Because any session or job with a particular session/job identity can access a :DATA file with the same session/job identity, you are urged to ensure that your session or job has a unique identity. You can help achieve this by assigning the session/job a session or job name along with the user and account names entered.

When a non-spooled devicefile containing a :DATA command is readied and recognized by MPE, it is placed in the READY state and can be allocated to a session or job without operator intervention. Prior to this access, however, the operator may delete this devicefile (with the =DELETE command) or abort the session or job (with the =ABORT command).

END-OF-FILE INDICATORS

The end-of-file indication is returned by the card reader and tape drivers under conditions specified by the initiators of read requests. The type of requests are as follows:

Type	Class of end-of-file
A	All records that begin with a colon (:).
B	All records that contain, starting in the first byte, :EOD, :EOJ, :JOB, and :DATA.
E	Hardware-sensed end-of-file (or an :EOF: command, on the card reader only).

In utilizing the card/tape devices as files via the File System, the following types are assigned:

File Specified	Type
\$STDIN	Type A.
\$STDINX	Type B.
Dev= CARD/TAPE	Type B, if device job/data accepting. Type E, if device not job/data accepting.

Any subsequent requests initiated to the driver following sensing of an end-of-file condition are rejected with an end-of-file indication.

When reading from an unlabeled tape file, the request encountering a tape mark responds with an end-of-file indication but succeeding requests continue to read data past the tape mark. Under these conditions, it is the responsibility of the calling program to protect against the occurrence of data beyond the end-of-file and to prevent reading off the end of the reel.

The HP 2894A Card Reader/Punch Unit does not provide a hardware end-of-file indication for input files. Therefore, if you plan to read data from this device when it is *not* configured to accept data via the :DATA command, you must ensure that your data deck is terminated by an MPE command that *simulates* the hardware end-of-file. This command is:

:EOF:

NOTE

The last colon in this command must be followed by a blank.

Records beginning with certain entries always automatically terminate the standard input files for sessions and jobs, or data files submitted via the :DATA command. These entries are listed in table 6-18.

Table 6-18. File-Terminating Entries

TYPE OF FILE	ENTRIES
Standard input files for sessions.	: followed by any other character. } Terminates \$STDIN.
	:EOD :EOF: } Terminate \$STDINX.
Standard input files for jobs.	:EOJ :JOB :EOD :DATA :EOF: } Terminate \$STDIN and \$STDINX.
	: followed by any other character. } Terminates \$STDIN. This record is then interpreted by the Command Interpreter as the next command to be executed.
:DATA File.	:EOD :JOB :DATA :EOF:

COMMAND/FILE-TYPE SUMMARY

The file commands described in this section and the types of devices to which they apply are summarized in table 6-19, below:

Table 6-19. Commands Vs. File Types

ALL DEVICES	DISC ONLY	DISC INPUT, TAPE OUTPUT	TAPE INPUT, DISC OUTPUT	ALL DEVICES EXCEPT DISC
:FILE :RESET	:ALTSEC :BUILD :LISTF :PURGE :RELEASE :RENAME :SAVE :SECURE	:STORE	:RESTORE	:DATA :EOF:

RUNNING SUBSYSTEMS AND USER PROGRAMS

SECTION

VII

MPE enables you to directly invoke various subsystems and user programs, thus allowing you to:

- Compile source-language programs, prepare them for execution, and run them.
- Perform on-line, interactive programming via the BASIC Interpreter.
- Run various other MPE subsystems and programs, such as the Editor and the File Copier.

These operations are discussed in this section.

NOTE

An MPE subsystem is any program that is run by a unique MPE command, such as :FORTRAN, :SPL, or :EDITOR.

COMPILING, PREPARING, AND EXECUTING PROGRAMS

Between the time you write a program and the time you receive its output from the system, the program undergoes the following steps, all under the direction of MPE:

1. Compilation
2. Preparation (Segmentation)
3. Allocation/Execution

In most cases, the details of these steps, described below, remain invisible to you during the normal flow of operations. When necessary, however, you can advance through each of these steps independently, completely controlling the specific details of each event along the way.

COMPILATION

Programs are entered into the computer in a source language, translated into binary form by a compiler, and stored on disc (figure 7-1A). Because the code making up a compiler is re-entrant, it can be shared by many users. Therefore, only one copy of a compiler is required in memory no matter how many programmers need it at one time to compile their programs.

MPE regards your source-language programs as consisting of program units. A *program unit* is a self-contained set of statements that is the smallest divisible part of any program or subprogram. For example, in FORTRAN, this is a main program, subroutine, function, or block data unit. In COBOL, this is a main program, subroutine, or section. In SPL, this is an outer block program unit or procedure. Thus, a *program* consists of one or more program units, one of which must be a main program (FORTRAN or COBOL) or an outer block (SPL). A *subprogram*, which is the smallest individually-compileable entity, in turn, consists of one or more program units.

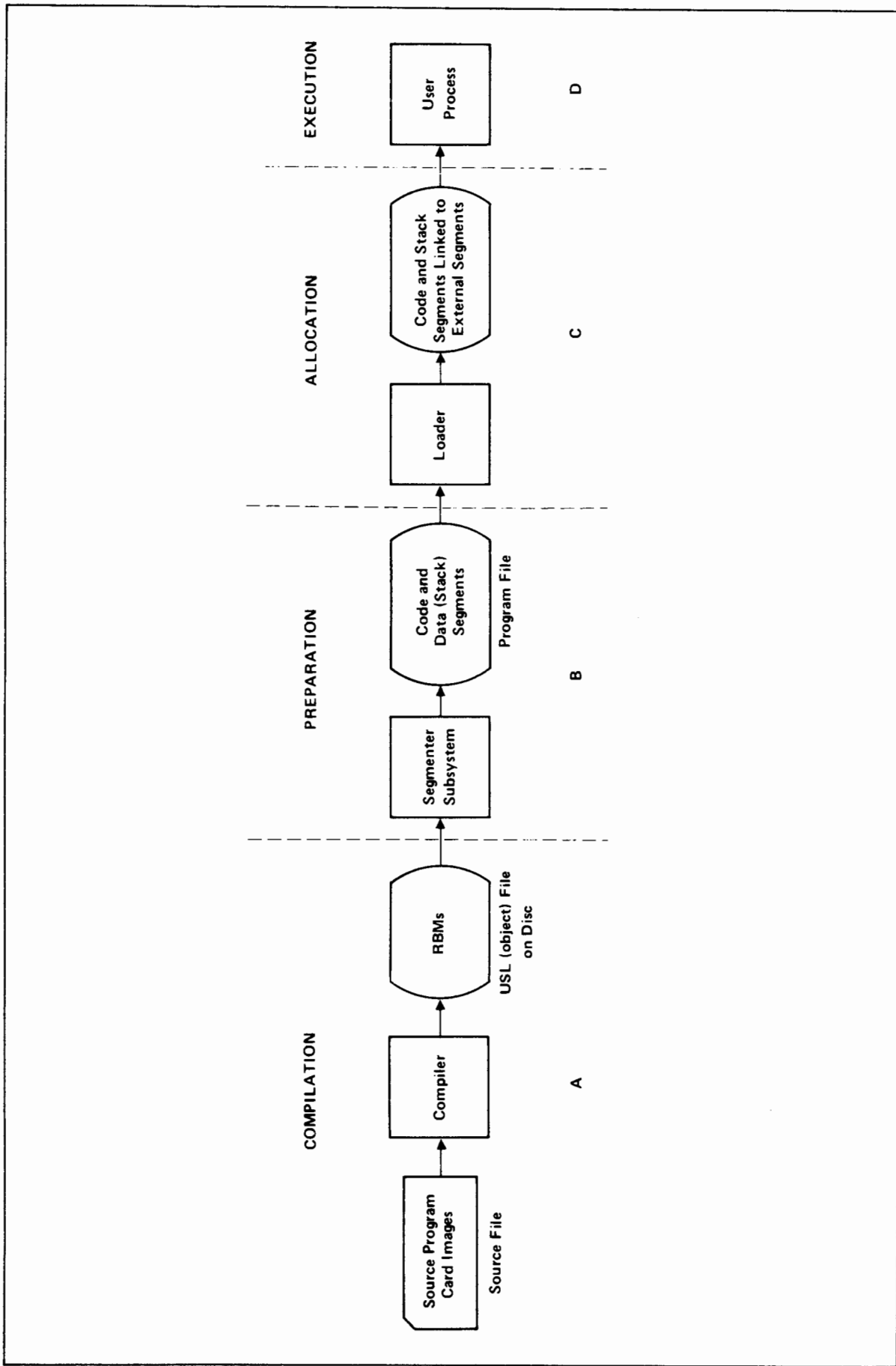


Figure 7-1. Program Management

When a source-language program is compiled, each program unit is transformed into a relocatable binary module (RBM) that contains user code plus header information that labels and describes this code. The RBMs are written onto a specially-formatted disc file called a user subprogram library (USL). There is one RBM for each program unit.

Over a period of time, you can produce RBMs on the same USL through several compilations. Thus, you can compile your main program and procedures separately. If there is one (and only one) main program or outer block RBM containing active entry points on the USL, it can be prepared and run.

During compilation, if a program unit is designated as privileged, the corresponding RBM is flagged as privileged in the USL.

PROGRAM PREPARATION

The USL resulting from compilation is not executable in its present state. Instead, it must be *prepared* for execution by the MPE Segmenter which binds the RBMs from the USL (object file) into linked, re-entrant code segments arranged in a *program file* (figure 7-1B). (In preparation, only *one* USL can be used for RBM input to the program file.) Each segment contains machine instructions produced from your program, plus linkages to other segments. The code in a segment cannot be altered because it is treated as read-only information; thus, it remains re-entrant and can be executed repeatedly by many users. When a program is prepared, a special data segment for the input of user data is also initially defined; this contains the stack.

During preparation, the USL and program files are opened as session/job temporary files. For both files, the Segmenter first searches for a file of the proper name that exists as a session/job temporary file; if such a file cannot be found, the Segmenter searches for a permanent file of the appropriate name. If no program file of the referenced name exists, the Segmenter creates a new program file that is saved in the session/job temporary file domain, and prepares into this. If any RBM from the USL is flagged as privileged, you must have the privileged mode (PM) optional capability *and* assign your program the privileged mode capability class in order to successfully prepare it; once prepared, the entire segment containing the privileged RBM is flagged as privileged. (During preparation, the capability-class attributes of the program are determined by yourself or by the default option supplied by MPE. The privileged mode capability class is assigned by entering the ;CAP= PM keyword parameter group in the :PREP or :PREPRUN command that requests preparation.)

ALLOCATION/EXECUTION

When a program is run, it is allocated and executed. In *allocation*, the MPE loader binds the segments from the program file to any referenced external segments from a segmented library (SL) (figure 7-1C). Then, the first code segment to be executed and the stack are moved into main memory. *Execution* now begins (figure 7-1D). (For further information about the details of execution, see Appendix B.)

Before allocation/execution, MPE checks to verify the following points:

- *For program files that are permanent files*, the capabilities assigned to the program must not exceed those assigned to the group to which the program file belongs; otherwise, an error occurs.

- *For program files that are temporary files in the session/job domain, the capabilities assigned to the program must not exceed those of the user running the program; otherwise, an error occurs.*
- *For privileged segments, when the NOPRIV parameter is omitted from the :RUN (program execution) command, the capabilities assigned to the program must include the privileged mode capability for the segment to be loaded in privileged mode; otherwise, an error occurs.*

The compilation, preparation, and execution of a program can be requested by individual commands or by a single command that performs all three operations. In subsequent pages, all of these commands are described.

NOTE

Users programming in SPL or running specialized applications may want to learn more about running programs (processes) and their code and data segments. This knowledge is particularly valuable for those who plan to modify areas in the data stack via some of the :PREP, :PREPRUN, and :RUN command parameters described in the Reference Specifications (Section II). For this type of information, see Appendix B of this manual, as well as the *General Information Manual* and *MPE Intrinsic Reference Manual*.

COMPILATION ONLY

If you want to compile a source-language program entered into the system through your standard input device, write the object code onto the standard default disc file used for compiler output, and transmit a listing of your program to your standard listing device, simply enter the name of the compiler in the form of an MPE command. The default file for object output is \$NEWPASS (if no passed file currently exists, or \$OLDPASS (if a passed file does exist). If the next command is one to prepare an object program for execution, \$NEWPASS can be passed to that command under the file name \$OLDPASS.

NOTE

A file can only be passed between commands or programs within the same session/job.

As an example, you can compile a COBOL program from the standard input file onto the default file for compilation, and save this file under the name COBFL for preparation during another job, by entering the following commands. The program is listed on the standard listing device:

:COBOL	<i>Compiles COBOL program.</i>
:SAVE \$OLDPASS, COBFL	<i>Saves object output on file COBFL.</i>

NOTE

Four compilers are available to translate source-language programs into USL form, as discussed above; these compilers are the COBOL, FORTRAN, RPG, and SPL compilers. An additional compiler, the BASIC compiler, is available to translate code generated by the BASIC Interpreter into USL form for faster execution; this compiler is discussed later in this section.

COMPILATION/PREPARATION

You may compile and prepare a source-language program by using a single MPE command, as indicated in table 7-1, below:

Table 7-1. Compilation/Preparation Commands

COMPILER	COMMAND
COBOL	:COBOLPREP
FORTRAN	:FORTPREP
RPG	:RPGPREP
SPL	:SPLPREP

If you wish, for instance, to compile an RPG program input from your standard input device, and then — without pausing — prepare it onto the standard default program file (\$NEWPASS), with the program listing transmitted to the standard listing device, enter:

```
:RPGPREP
```

The USL file created during compilation is a temporary file passed directly to the preparation mechanism; you can access it, but only under the name \$OLDPASS, and only if the program was not prepared into the file named \$NEWPASS. Therefore, if you want to save the compiled USL, you must specify some other file as the second positional parameter in this command, and then use a :SAVE command. For instance:

*Note first positional parameter omitted;
textfile read from standard input device.*

```
:RPGPREP , COMFL  
:SAVE $OLDPASS, NUSL
```

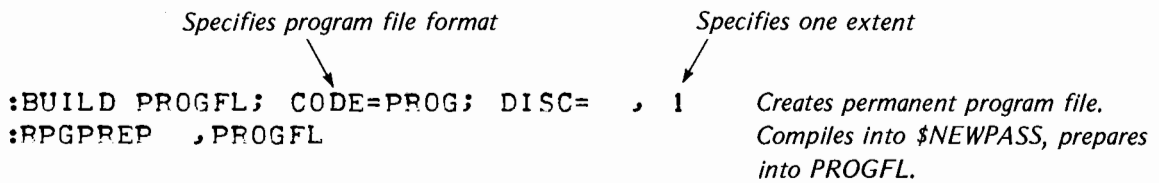
*Compiles into \$NEWPASS, prepares into COMFL.
Saves USL, changing file name from \$OLDPASS
(formerly \$NEWPASS) to NUSL.*

The program file is also created as a temporary file (unless you prepare the program onto a previously-created file). Therefore, if you want to save a new program file, you may use the :SAVE command as follows:

```
:RPGPREP , COMFL  
:SAVE $OLDPASS, NUSL  
:SAVE COMFL
```

*Compiles into \$NEWPASS, prepares into COMFL.
Saves USL under name NUSL.
Saves program file named COMFL.*

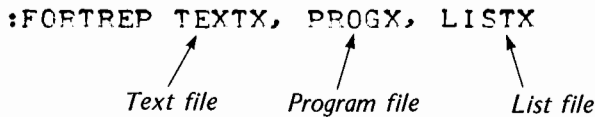
Alternatively, if you want to create a new permanent file into which to prepare your program, you may use the :BUILD command. When you do this, however, you must specify a *filecode* of PROG (or 1029) to specify program file format, and limit this file to one extent.



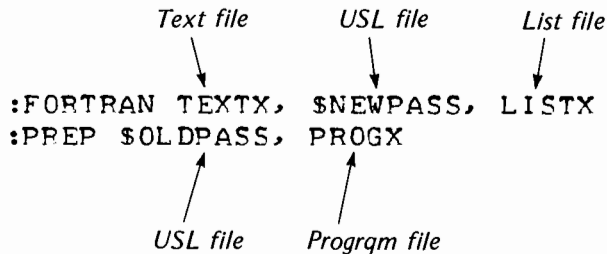
Suppose you are receiving your input file from a device other than the standard input device, for instance, from a disc file created via the Editor. Suppose, also that you wish to transmit your listing to a device other than the standard listing device — for instance, to another line printer. You could do this as follows, specifying additional positional parameters in the compile/preparation command. For instance:



The compilation/preparation command effectively merges the compilation and preparation steps into one continuous function. For instance, the command:



is equivalent to



NOTE

The :PREP command, to prepare a previously-compiled program, is discussed later in this section.

COMPILED/PREPARATION/EXECUTION

You may also compile, prepare, and execute a source-language program via a single MPE command, as shown in table 7-2, below.

Table 7-2. Compilation/Preparation/Execution Commands

COMPILER	COMMAND
COBOL	:COBOLGO
FORTRAN	:FORTGO
RPG	:RPGGO
SPL	:SPLGO

To compile, prepare, and execute a FORTRAN program, input from your standard input device with listing output produced on your standard listing device, for instance, simply enter:

```
:FORTGO
```

The USL file created during compilation is a temporary file passed directly from the compiler to the Segmenter; you cannot access this file. The program file created during preparation is also a temporary file, passed directly to the execution mechanism. You can access it under the file name \$OLDPASS. To save this file under the name MYPROG, for instance, enter:

```
:FORTGO                               Compiles, prepares, and executes program.
:SAVE $OLDPASS, MYPROG                Saves program file ($OLDPASS) under the name MYPROG.
```

To enter your source input from a device other than your standard input device, and/or direct the listing to a device other than your standard listing device, simply name the input and listing files as command parameters:

```
:FILE MTAPE; DEV=TAPE ← Identifies MTAPE as a magnetic tape file.
:FILE PRTR; DEV=FASTPRTR ← Identifies PRTR as a special line-printer file
                          { (FASTPRTR class name).
:FORTGO *MTAPE, *PRTR ← Compiles from MTAPE, writes listing to PRTR,
                        ↑      ↑
                        Text file List file
                        prepares, and executes.
```

The compilation/preparation/execution command effectively combines these three steps into one continuous operation. For instance, the command:

```
:COBOLGO TEXTFL, LISTFL
      ↑      ↑
      Text file List file
```

is equivalent to:

```
      Text file      USL file      List file
      ↓              ↓              ↓
:COBOL TEXTFL, $NEWPASS, LISTFL
:PREP $OLDPASS, $NEWPASS
:RUN $OLDPASS
      ↑              ↑              ↑
      Program file  USL file      Program file
```

NOTE

The :RUN command, to allocate/execute a previously-prepared program, is discussed later in this section.

PREPARATION ONLY

If a source program has been compiled onto a USL, you can prepare it for execution with the :PREP command. Unless you prepare the program onto a previously-created program file, this command will create a program file of the appropriate format for you in the session/job temporary file domain. In fact, it is recommended that you specify a non-existent program file in the :PREP command; this allows MPE to create a file of optimum size and characteristics. The following command, which prepares a program from the USL file USLX (on disc) to the program file PROGX, creates such a file.

```
      Non-existent file
      ↓
:PREP USLX, PROGX      Prepares program onto program file.
:SAVE PROGX           Saves program file.
```

You can create a program file in the permanent file domain by using the :BUILD command. When you do this, you must be sure to specify a *filecode* of PROG (or 1029) for this file, and to limit the file to one extent; program files are not allowed to span more than one extent. The following :BUILD command creates such a file, which is then used by the :PREP command below:

```
:BUILD PFILE; CODE=PROG; DISC= , 1
:PREP UFILE, PFILE
```

You can obtain a listing describing the prepared program by specifying the PMAP keyword parameter in the :PREP command. For example,

```
:PREP USLX, SCR4; PMAP
```

The listing appears in the format shown in figure 7-2; on this listing, significant entries are indicated with arrows, keyed to the items in table 7-3. All numbers in the listing except Item 13 (elapsed time) and Item 20 (central processor time) are octal values. Some of the terms appearing in this table apply to elements beyond the scope of this discussion. For further clarification, see Appendix B of this manual, the *General Information Manual*, and *MPE Segmenter Reference Manual*.

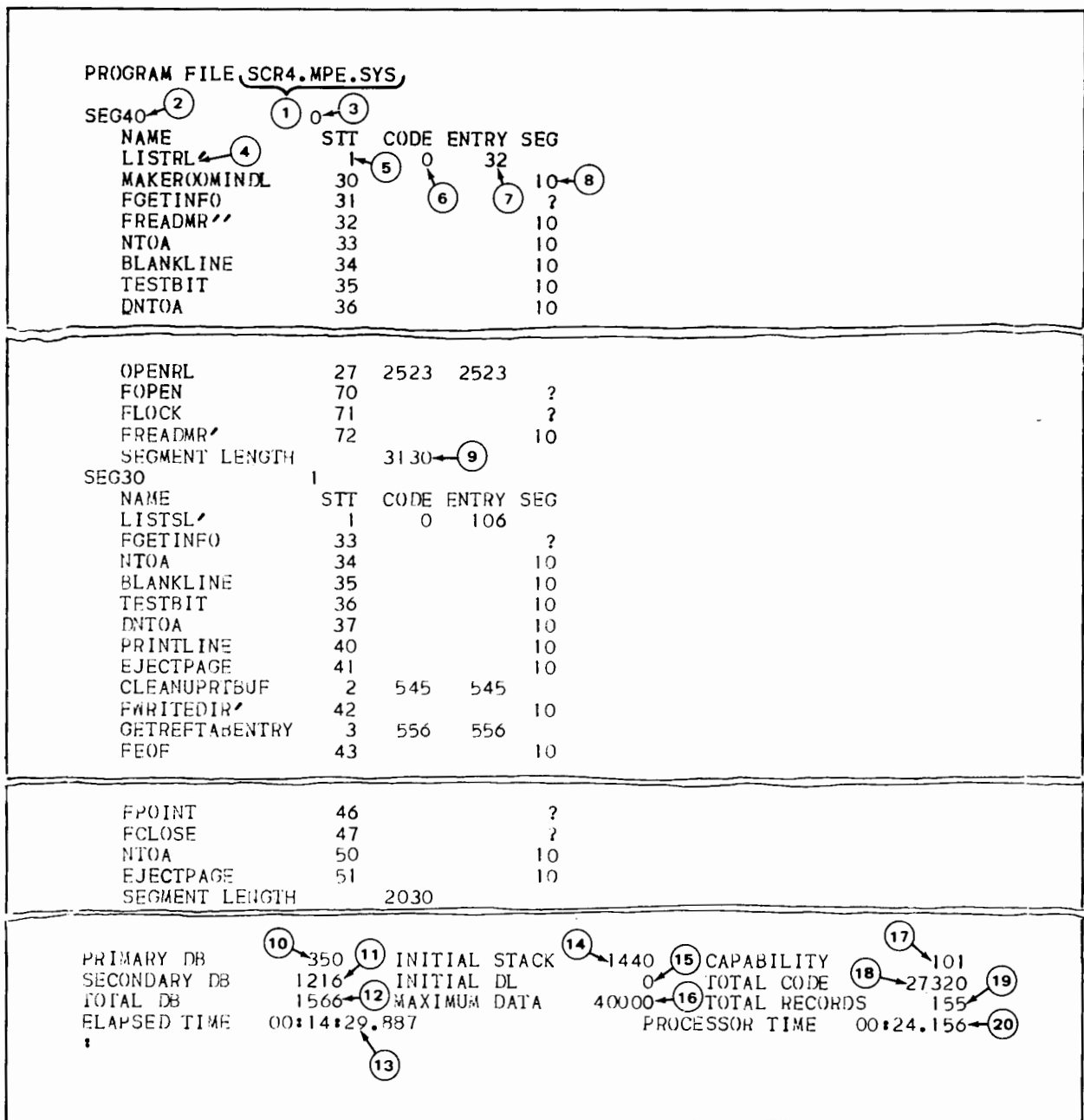


Figure 7-2. Listing of Prepared Program

When you prepare a program file using the :PREP command, you can request the Segmenter to search a relocatable library file (RL) for external procedures required by your program. In the RL, these procedures exist in RBM form as they do in a USL; after preparation, they are stored in a segment and linked to the program file. The following example shows how to request a search of the relocatable library named RL02 in the Public Group of the System Account.

```

:FILE RL02 = RL02.PUB.SYS
:PREP MYUSL, MYPROG; RL = *RL02

```

Requests search of RL.

Many other preparation parameters can be specified with the :PREP command; see the :PREP command reference specification. To understand how to use those parameters that modify areas in the data stack, you should also read the information in Appendix B of this manual.

Table 7-3. Prepared Program Listing Key

Item No.	Meaning
1	The name of the program file (filename.groupname.accountname).
2	The segment name.
3	The (logical) segment number.
4	The program unit entry-point name or external procedure name.
5	The assigned entry number in the Segment Transfer Table (STT).
6	The beginning location of the procedure code in the segment.
7	The location of the entry point in this segment.
8	The (logical) segment number of the segment containing this <i>external</i> procedure. If this entry is a number, then the procedure is external to the segment but internal to the program file; if it contains a question mark (?), then the procedure is external to the segment <i>and</i> external to the program file.
9	The segment length (in words).
10	The primary DB area size.
11	The secondary DB area size.
12	The total DB area size.
13	The time elapsed during preparation process.
14	The initial stack size.
15	The initial DL size.
16	The maximum area available for data (maximum Z-DL size).
17	Capability of program file.
18	Total code in file.
19	Total records in file.
20	Total central processor time used during preparation process.

PREPARATION/EXECUTION

If a source program has been compiled on a USL file, you can also prepare it for execution and then execute it immediately via the :PREPRUN command. (The Segmenter prepares the program into the program file \$NEWPASS, and the Loader then executes \$OLDPASS.) As an example, to prepare a program from the USL file XUSL, execute it, and save the program file under the name XPROG, enter:

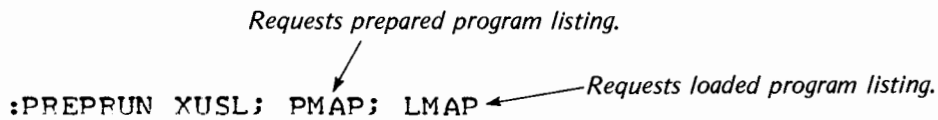
```
:PREPRUN XUSL
:SAVE $OLDPASS, XPROG
```

As with the `:PREP` command, you can obtain a descriptive listing of the prepared program by specifying the `PMAP` keyword parameter. You can also request a listing of the allocated (loaded) program by specifying the `LMAP` keyword parameter. For example:

Requests prepared program listing.

```
:PREPRUN XUSL; PMAP; LMAP
```

Requests loaded program listing.



The listing of the loaded program appears in the format shown in figure 7-3; on this listing, significant entries are indicated with arrows, keyed to items appearing in table 7-4. This listing shows the externals referenced by the program and the segments from segmented libraries to which they were bound.

Many other parameters can be specified with the `:PREPRUN` command; see the reference specification for this command. To understand how to use those parameters that modify areas in the data stack, you should also read the information in Appendix B of this manual.

EXECUTION ONLY

If a program has been prepared on a program file, you can run it by entering the `:RUN` command. For example, to run the program residing on the program file `XLAB`, enter:

```
:RUN XLAB
```

This command permits you to specify entry points other than the primary entry point for beginning the program. For instance, to run the program `XLAB` beginning at the secondary entry point `SECPT`, enter:

```
:RUN XLAB, SECPT
```

With this command, you can also request a listing of the loaded program, as follows:

```
:RUN XLAB; LMAP
```

Many other parameters can be specified with the `:RUN` command; see the reference specification for this command. To understand how to use those parameters that modify areas in the data stack, you should also read Appendix B of this manual.

Table 7-4. Loaded Program Listing Key

Item No.	Meaning
1	The name of the program file (filename.groupname.accountname).
2	The name of the external procedure.
3	The type of the segment referencing the external procedure, where: PROG = program segment. GSL = group segmented library segment. PSL = public segmented library segment.
4	External parameter checking level.
5	External segment transfer table (STT) number.
6	External (logical) segment number.
7	Entry point segment type, where: GSL = group segmented library segment. PSL = public segmented library segment. SSL = system segmented library segment.
8	Entry point parameter checking level.
9	Entry point segment transfer table (STT) number.
10	Entry point (logical) segment number
11	A list of the code segment table (CST) numbers to which the program file segments were assigned. The list is ordered by logical segment number.

USING THE BASIC INTERPRETER

The BASIC Interpreter is generally used for on-line programming during sessions. However, you can also use it to interpret BASIC programs submitted in batch-job mode. In either case, you call the BASIC Interpreter with the :BASIC command. For instance, to enter BASIC commands and statements as well as input data from your terminal, with program listing and output also directed to the same terminal, you simply run the interpreter as follows:

```
:BASIC
```

In certain cases, you may not wish to use the terminal for input/output. For instance, you may wish to enter your BASIC commands and statements from a disc file, read data from another disc file, and transmit your program listing and output to a line printer. In a session, you can do this as follows. (In this example, the commands and statements reside on the disc file BASP, the input data resides on the disc file BASD, and the listing and output is sent to the printer file BASL.)

```
:FILE BASL; DEV=LP          Defines BASL as line-printer file.
:BASIC BASP, BASD, *BASL    Invokes BASIC.
```

USING THE BASIC COMPILER

Once you have written a program using the BASIC Interpreter and debugged that program, you may wish to run the program with the shortest execution time possible. You can do this with the aid of the BASIC compiler, by following these steps:

1. Use the BASIC Interpreter command `SAVE filename, FAST` to store the program into a permanent disc file in special SAVE FAST format.
2. Now, run the BASIC compiler and enter the compiler command `$COMPILE` to compile the program into USL format so that it exists in the system in RBM form rather than as data in a data file. When in RBM form, programs can generally be prepared and executed in shorter time than they can be run in the form produced by the BASIC interpreter.
3. Prepare and run the program via the MPE `:PREP` and `:RUN` commands or MPE `:PREPRUN` command.

As an example, suppose you have stored your BASIC program on the SAVEFAST file named MYRUN on disc. You run the BASIC compiler by entering the `:BASICOMP` command from your terminal. You then compile the program with the `$COMPILE` command and exit from the compiler with the `$EXIT` command, also entered from your terminal. Next, you prepare and run the program with the `:PREPRUN` command:

```

      .
      .
      .
: BASICOMP ← Runs BASIC compiler, accepting commands from $STDINX,
              and requesting $NEWPASS/$OLDPASS for RBM output
              and $STDLIST for listing output.
$COMPILE MYRUN ← Compiles from SAVE FAST file named MYRUN
                  onto USL named $OLDPASS.
$EXIT ← Exits from BASIC compiler.
:PREPRUN $OLDPASS ← Prepares and runs program.
      .
      .
      .
```

In the `:BASICOMP` command, the default assignments for the compiler command file, USL file, and listing output file, are `$STDINX`, `$NEWPASS/$OLDPASS`, and `$STDLIST`, respectively. As command parameters, however, you can specify different input/output files. (See the reference specification for the `:BASICOMP` command.)

In addition to the `:BASICOMP` command, the BASIC compiler provides the `:BASICPREP` command to compile and prepare a BASIC program, and the `:BASICGO` command, to compile, prepare, and execute a BASIC program. (See the reference specifications for these commands.)

USING OTHER SUBSYSTEMS AND PROGRAMS

In addition to the compilers noted above and the BASIC Interpreter, you can also execute other HP-designed subsystems and programs that run under MPE. The subsystems are called by entering unique MPE commands. These include the Editor (`:EDITOR` command), the Segmenter (`:SEGMENTER` command), the HP 2780/3780 Emulator (`:RJE` command), and the Debug Facility (`:DEBUG`

command, which requires privileged mode capability); the reference specifications for these commands appear in Section II of this manual as well as in the appropriate subsystem manuals. The other programs available through HP are executed through the general MPE execution command, :RUN. These programs include the Sort/Merge Facility (SORT), the File Copier (FCOPY), and the Stand-Alone Diagnostic Utility Program (SDUP); the command formats for running these programs are discussed in the appropriate program manuals.

IMPLICIT :FILE COMMANDS FOR SUBSYSTEMS

Compilers and other subsystems that run under MPE accept actual file designators as parameters in the commands that call these programs. Although you are not generally aware of this fact, when an actual file designator appears as a command parameter, it is *automatically* equated to a formal file designator, used within the subsystem, by an *implicit :FILE command* issued by the command executor. For instance, within the FORTRAN compiler, the formal file designator for the *textfile* input is FTNTEXT. In the :FORTRAN command format, this is related to the *textfile* parameter as shown below:

Specifies textfile input.
↓

```
:FORTRAN [textfile] [, [uslfile] [, [listfile] [, [masterfile] [, [newfile] ] ] ]
```

When you specify a file named ALSFILE for *textfile* input as noted below,

```
:FORTRAN ALSFILE
```

MPE implicitly issues the following :FILE command, invisible to yourself:

```
:FILE FTNTEXT=ALSFILE
```

When calling a compiler or subsystem, any valid actual file designators can be used as command parameters, including those of the **formaldesignator* (back-reference) format.

As another example of an implicit :FILE command, suppose you specify a file on magnetic tape used as a source file during a FORTRAN compilation:

```
:FILE SOURCE=TAPE1,OLD; DEV=TAPE; REC=-80  
:FORTRAN *SOURCE
```

When these commands are encountered, the compiler executor issues the following implicit :FILE command, back-referencing your previous :FILE command:

```
:FILE FTNTEXT=*SOURCE
```

When a compiler or subsystem terminates, MPE implicitly resets each formal designator previously set by an implicit :FILE command issued as a result of the compiler/subsystem call. This minimizes confusion between the subsystem's designators and the ones you use.

You can avoid issuing :FILE commands that conflict with other pre-defined :FILE commands if you know the formal file designators used by MPE subsystems and commands. The formal file designators for compilers, interpreter, and other subsystem commands are listed in table 7-5.

Table 7-5. Subsystem Formal File Designators

Command	Parameters	Formal File Designator
:BASIC	<i>commandfile</i> <i>inputfile</i> <i>listfile</i>	BASCOM BASIN BASLIST
:BASICOMP	<i>textfile</i> <i>uslfile</i> <i>listfile</i>	BSCTEXT BSCUSL BSCLIST
:FORTRAN	<i>textfile</i> <i>uslfile</i> <i>listfile</i> <i>masterfile</i> <i>newfile</i>	FTNTEXT FTNUSL FTNLIST FTNMAST FTNNEW
:SPL	<i>textfile</i> <i>uslfile</i> <i>listfile</i> <i>masterfile</i> <i>newfile</i>	SPLTEXT SPLUSL SPLLIST SPLMAST SPLNEW
:COBOL	<i>textfile</i> <i>uslfile</i> <i>listfile</i> <i>masterfile</i> <i>newfile</i>	COBTEXT COBUSL COBLIST COBMAST COBNEW
:RPG	<i>textfile</i> <i>uslfile</i> <i>listfile</i> <i>masterfile</i> <i>newfile</i>	RPGTEXT RPGUSL RPGLIST RPGMAST RPGNEW
:BASICPREP	<i>textfile</i> <i>progfile</i> <i>listfile</i>	BSCTEXT BSCPROG BSCLIST
:FORTPREP	<i>textfile</i> <i>progfile</i> <i>listfile</i> <i>masterfile</i> <i>newfile</i>	FTNTEXT FTNPROG FTNLIST FTNMAST FTNNEW

Table 7-5. Subsystem Formal File Designators (continued)

Command	Parameters	Formal File Designator
:SPLPREP	<i>textfile</i> <i>progfile</i> <i>listfile</i> <i>masterfile</i> <i>newfile</i>	SPLTEXT SPLPROG SPLLIST SPLMAST SPLNEW
:COBOLPREP	<i>textfile</i> <i>progfile</i> <i>listfile</i> <i>masterfile</i> <i>newfile</i>	COBTEXT COBPROG COBLIST COBMAST COBNEW
:RPGPREP	<i>textfile</i> <i>progfile</i> <i>listfile</i> <i>masterfile</i> <i>newfile</i>	RPGTEXT RPGPROG RPGLIST RPGMAST RPGNEW
:BASICGO	<i>textfile</i> <i>listfile</i>	BSCTEXT BSCLIST
:FORTGO	<i>textfile</i> <i>listfile</i> <i>masterfile</i> <i>newfile</i>	FTNTEXT FTNLIST FTNMAST FTNNEW
:SPLGO	<i>textfile</i> <i>listfile</i> <i>masterfile</i> <i>newfile</i>	SPLTEXT SPLLIST SPLMAST SPLNEW
:COBOLGO	<i>textfile</i> <i>listfile</i> <i>masterfile</i> <i>newfile</i>	COBTEXT COBLIST COBMAST COBNEW
:RPGGO	<i>textfile</i> <i>listfile</i> <i>masterfile</i> <i>newfile</i>	RPGTEXT RPGLIST RPGMAST RPGNEW
:EDITOR	<i>listfile</i>	EDTLIST
:SEGMENTER	<i>listfile</i>	SEGLIST
:RJE	<i>commandfile</i> <i>inputfile</i> <i>listfile</i> <i>punchfile</i>	RJECOM RJEIN RJELIST RJEPUNCH

The formal file designators for optional parameters in the commands that prepare and run programs and store and restore files are shown in table 7-6. Implicit :FILE commands are not issued for these designators. Rather, the formal designator specifies the destination of listings generated by the command executor if you do not re-specify the destination by issuing an *explicit* :FILE command. (This :FILE command, if issued, remains in effect throughout the session/job, unless an applicable :RESET command or another :FILE command is issued.)

Table 7-6. Preparation/Execution and Store/Restore Command Formal File Designators

Command	Parameters	Formal File Designator
:PREP :PREPRUN } }	P MAP	SEGLIST
:PREPRUN :RUN } }	L MAP	LOADLIST
:RESTORE :STORE } }	S H OW	S Y S L I S T

DETERMINING DEVICE AND DEVICEFILE STATUS

SECTION

VIII

MPE allows the input/output of information via many different types of hardware devices; as noted in Section VI, these devices are divided into two general classifications — discs and devices used for devicefiles. In this section, devices and devicefiles are discussed with respect to their operational states.

HOW MPE HANDLES DEVICES

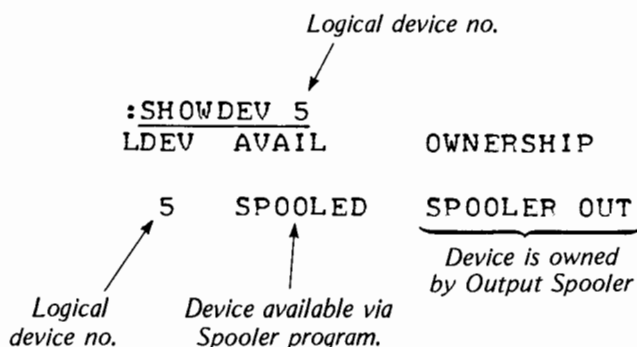
MPE recognizes devices by their device class names and logical device numbers, assigned during system configuration. The class name may refer to a unique device or to a group of several devices. The logical device number, however, is unique for any particular device.

During configuration, the System Supervisor or Console Operator can determine whether a device may be used to initiate sessions (via the :HELLO command) or batch jobs (via a :JOB command), or to accept data made available to sessions/jobs (via the :DATA command), or whether the device may be used for all of these functions or none of them.

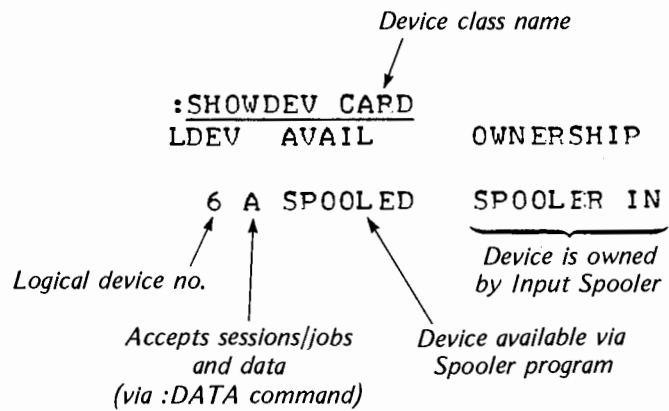
While MPE is operational, any particular device may or may not be available to you for use. For instance, discs are always available, as are unit-record devices not being used for direct or spooled input or output. Devices other than discs being used by another session or job, however, are not available to you.

Devices are also subject to "ownership". For example, a device is said to be owned by a session or job if it is being used for input/output by that session/job. A device can also be owned by an MPE Spooler program if data is being transferred between that device and disc. A device can be owned by a diagnostic testing program if the Console Operator has allocated the device to someone running such a test. Device ownership and availability are directly related, since devices owned by certain running programs (processes) are not available for general use.

Occasionally, you may need to determine the status of some particular device. You may, for example, wish to ensure that a device is connected on-line and that it is not presently owned by some other session/job before you direct listing output to that device. You can do this by using the :SHOWDEV command, referencing the logical device number of the device. For example,



You may also wish to report the status of all devices belonging to a particular device class (assigned a certain device class name). You can do this by referencing the class name in place of the logical device number in the :SHOWDEV command:



You can also display the status of all devices on the system. To do this, enter simply:

```

:SHOWDEV
LDEV  AVAIL  OWNERSHIP

  1  DISC    17 FILES
  4  SPOOLED SPoolER OUT
  5  SPOOLED SPoolER OUT
  6 A SPOOLED SPoolER IN ← Device owned by Input Spooler
  7  UNAVAIL #J19: 1 FILES ← Device owned by job
  8  AVAIL
  9  AVAIL
10 A AVAIL
11  AVAIL
12  AVAIL
13  AVAIL
14 A AVAIL
16  DISC    76 FILES ← Disc file with 76 extents
17  DISC    35 FILES
20 A AVAIL
21 A AVAIL
22 A AVAIL
23 A AVAIL
24 A AVAIL
25 A AVAIL
26 A UNAVAIL #S18: 2 FILES
27 A AVAIL
28 A AVAIL
29 A AVAIL
30 A AVAIL
31 A AVAIL
32 A AVAIL
33 A AVAIL
34 A UNAVAIL #S26: 2 FILES ← Device owned by session
35 A AVAIL
40 A AVAIL
41 A AVAIL
42 A UNAVAIL #S28: 6 FILES
43 A UNAVAIL #S37: 2 FILES
44 A AVAIL
45 A AVAIL
46 A AVAIL
47 A AVAIL
48 A AVAIL
49 A AVAIL
50 A AVAIL
51 A UNAVAIL #S17: 6 FILES
52 A AVAIL
53 A AVAIL
54 A AVAIL
55 A AVAIL

```

HOW MPE HANDLES DEVICEFILES

As noted in Section VI, a *devicefile* is a file currently being input to or output from any peripheral device except a disc. When information exists on such a device but is not being processed, MPE cannot recognize it as a file. Thus, information on cards is not identified as a file until the cards are loaded into the card reader and reading begins; data being written to a line printer is no longer regarded as a file when output to the printer terminates. A devicefile is accessed exclusively by the session or job that acquires it, and is *owned* by that session/job until the session/job explicitly releases it or terminates.

NOTE

Spooled devicefiles, although temporarily residing on disc, are considered *devicefiles* in the fullest sense because they are always originated on or destined for devices other than disc, and because you generally remain unaware of their storage on disc as an intermediate step in the spooling process. Whether they deal with spooled or unspooled devicefiles, your programs handle input/output as though the files reside on non-disc devices. The Console Operator, not the user, controls the spooling operation.

A devicefile is identified by the unique logical device number of the device on which it originated (input file) or is destined for (output file), or by the device class name assigned to that device (if this device is the only one belonging to this class). In addition, MPE assigns to each devicefile a unique identifier called the *devicefileid*. For input files, the *devicefileid* takes the form *#Innn*; for output files, it appears in the format *#Onnn*. An idle device is known by its logical device number or device class name only; but an active device (being used for a devicefile) has a *devicefileid*. In order for the system to keep track of who is using the device (who created the devicefile), additional information is also recorded:

- Session/Job Number, assigned by MPE at log-on time.
- File Name, as defined by user or system.
- User Name, when supplied in :DATA command only.
- Account Name, when supplied in :DATA command only.
- Logical device number of the device.
- State (OPENED, READY, ACTIVE or LOCKED).

INPUT DEVICEFILES

Input devicefiles are input files submitted on non-disc devices. There are three types of input devicefiles:

- Session/job input devicefiles.
- Data input devicefiles.
- Operator-assigned input devicefiles.

SESSION/JOB INPUT DEVICEFILES. A :HELLO or :JOB command entered on a device that is configured to accept sessions or jobs causes the creation of a devicefile on behalf of that session or job.

This file always assumes the system-defined name \$STDIN[X]. Such a devicefile is always automatically recognized on session/job-accepting devices, and is de-allocated at session/job termination; that is, the input devices always attempt to read the first command automatically.

DATA DEVICEFILES. The :DATA command (Section VI) provides you with a method of associating a data devicefile with a specific *user.account* name and optionally, a filename. :DATA-accepting devices, like session/job-accepting devices, always attempt to read the first command automatically. MPE scans the :DATA command for validity. If the command is syntactically correct, and the user and account specified in the command are present on the system, the devicefile is created. At this point, the specified user could access this devicefile without operator intervention, but only by accessing the devicefile via the *user.account name* and optional *filename* specified in the :DATA command; in this way, a session/job is allocated the devicefile upon its initial request for the devicefile. This mechanism simplifies the task of trying to keep track of what data belongs to what user. When the :DATA-accepting device is spooled, the system automatically reads the first command (:DATA) as usual. However, if the :DATA command is valid, the entire devicefile is spooled to the disc where it resides until accessed by the creating user (:DATA *username.accountname;filename*) or is deleted by the Console Operator.

OPERATOR-ASSIGNED DEVICEFILES. Input devices that do not accept :HELLO, :JOB, or :DATA commands do not automatically recognize data. When you request input data from a non-accepting device, the Console Operator is automatically asked to assign the device for your exclusive use. Files on such devices are called *operator-assigned devicefiles*; they begin simply with your first data record. Command images that have a colon as the first character may be read on such "non-accepting" devices; this is not true, however, of devices that accept :HELLO, :JOB, or :DATA commands, which treat a colon in Column 1 as an end-of-file indication. Operator-assigned devicefiles cannot be spooled.

ACCESSING DEVICEFILES. Your programs access input data by opening the input devicefile. Since the Console Operator normally controls the device whereas you control the device-opening program, some co-ordination or cooperation between you and the Operator is required. If the device to be used is configured as :DATA-accepting, the appropriate :DATA and :EOD (file terminating) commands are added to the data as the first and last records respectively. The complete data file is placed in the device and the system automatically reads the :DATA command. The devicefile is now known to the system and can be accessed by your program.

If your program is executed prior to the reading of the :DATA command, the devicefile is not known to the system and a message appears on the system console asking the Operator to assign a device for your request. In this case, the Operator places the file in the appropriate device as before, and after the :DATA command is read, he enters a =REPLY command to direct the accessing program to scan for the devicefile again, and open it.

If the device you select is not the :DATA-accepting type, the Operator must allocate the device when your program requests it.

A program waiting for an Operator's reply or action remains suspended indefinitely. If for some reason the device requested is unavailable or does not exist, the Operator can reply with an assigned logical device number of zero; this causes the devicefile-opening operation to fail. This failure requires you to make the necessary corrections to your program or :FILE command and run the program again.

INPUT SPOOLING. Under Console-Operator control, spooling buffers entire devicefiles onto disc (as described in Sections III and IV). User access to a devicefile is directed automatically to the disc copy, which is called a *spooled devicefile* (or sometimes, a *spoolfile*). Since a disc is a sharable device, jobs can concurrently access multiple spoolfiles which came from, or are destined for, the same non-sharable device. Furthermore, the copying operation between disc and a non-sharable device is designed to achieve high utilization of the non-sharable device.

MPE's spooling facility is invisible to users. This means that your sessions/jobs are unaware that they are accessing spoolfiles instead of real devicefiles on non-sharable devices. Specifically, there is no difference in programming for the access of a spooled device as opposed to a real device (although a few exceptional returns exist for programs accessing hardware functions such as reading hardware status).

Certain devices are spooled automatically when MPE is cold-loaded. The Console Operator can also initiate spooling on an unowned, non-sharable device. He can also terminate spooling and return such a device to the unowned state. Input spoolers may be started on those card readers or magnetic tape units designated as job-accepting devices.

When a non-sharable session/job- or :DATA-accepting input device is spooled, the device (called the *spooler*) is acquired by (belongs to) a spooling program. This program (process) is activated when the spooling operation is enabled. The purpose of the program is to perform read operations from the device (spooler) and copy or write this input data to a disc file called the ACTIVE file. Reading from the input device, and writing to the ACTIVE file continues, record by record, until the logical end-of-file indication is encountered. At this time, the ACTIVE file on the spooling disc becomes a READY file, and as such is available for access by programs or MPE. A program that opens a spooled input devicefile gains access to the newly-created READY file, at which time this file assumes the OPENED state. When the program closes the file, the file is deleted from the system. The normal transition for spooled input devicefiles is:

1. ACTIVE: In the process of being created but not yet complete.
2. READY: Ready for access by a program or deletion by the Console Operator.
3. OPENED: Being accessed by a user program or by MPE (being read).

This transition is illustrated in figure 8-1.

To summarize, the operation of the input spooler is to continually attempt to read data from the input device. An ACTIVE file is created when a :JOB command (from a job-accepting spooler) or :DATA command (from a :DATA-accepting spooler) is encountered in the input device stream. The input spooler validates the :JOB or :DATA command and copies subsequent records into the ACTIVE spooled devicefile. When the spooler encounters an :EOD, :EOJ, :JOB, or :DATA command or a physical end-of-file, the ACTIVE spooled devicefile becomes a READY file. READY spooled input devicefiles are thus available for access by user programs (via :DATA command) or by MPE (via :JOB command). When there are no more devicefiles on the device, the spooler enters a WAITING state, awaiting more input.

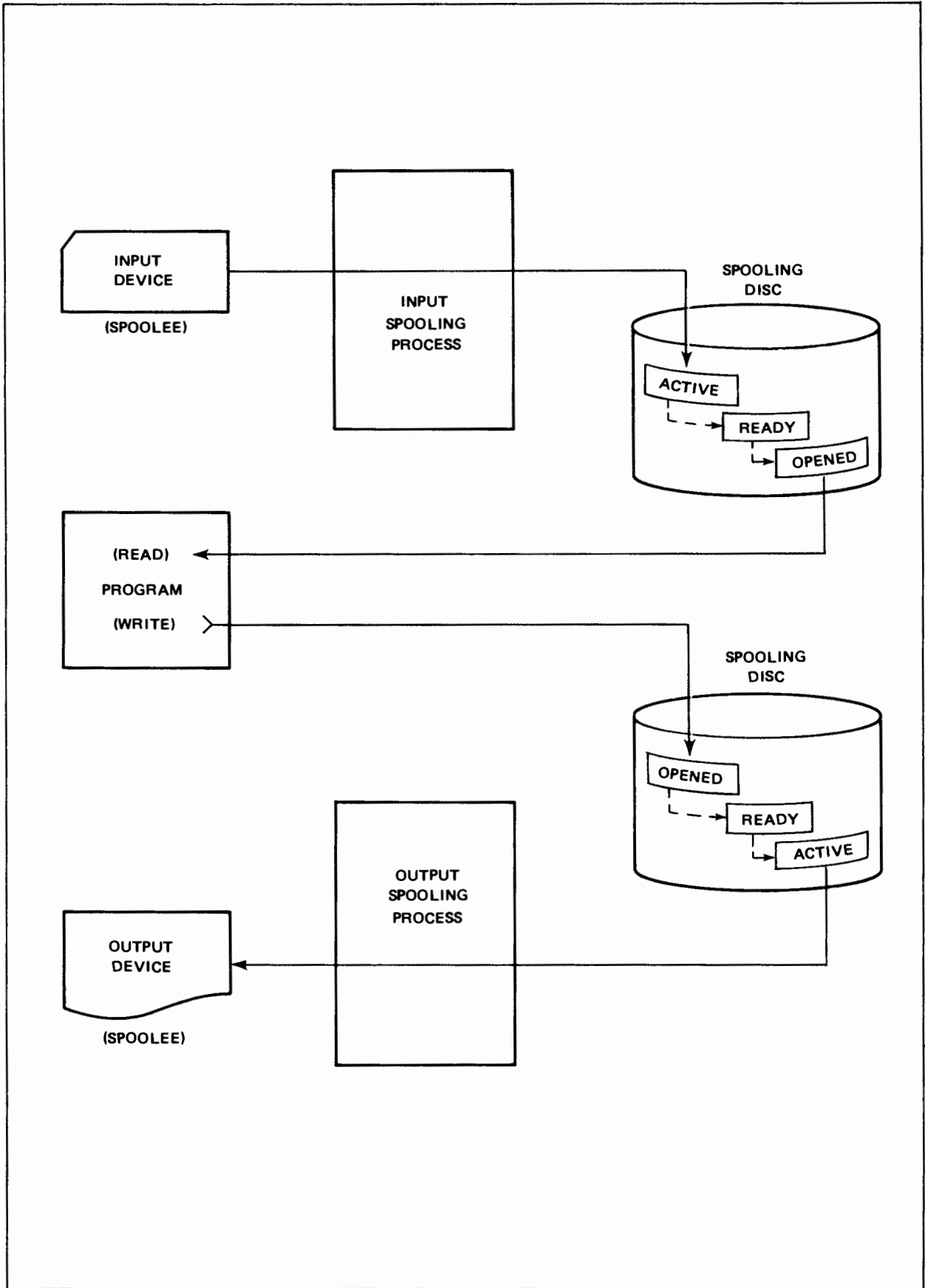
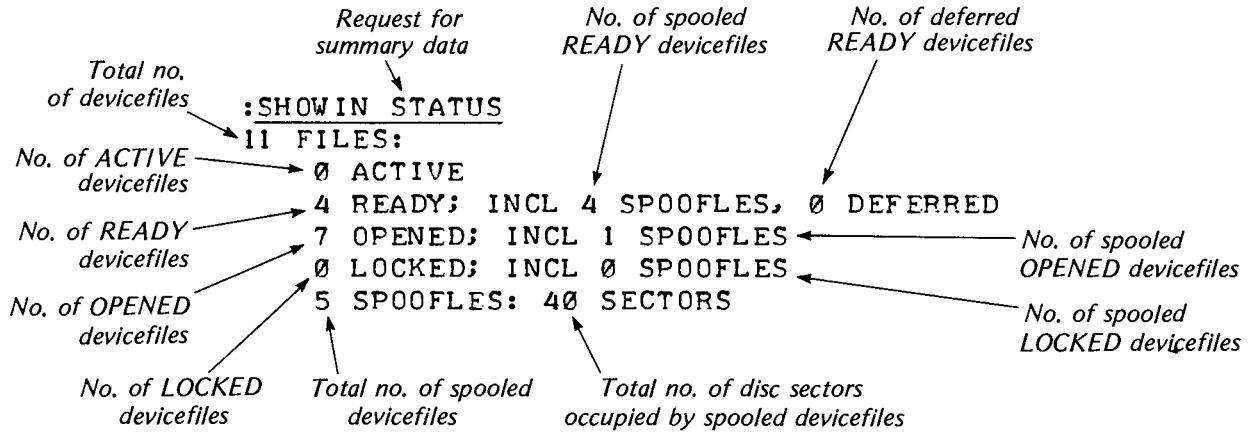
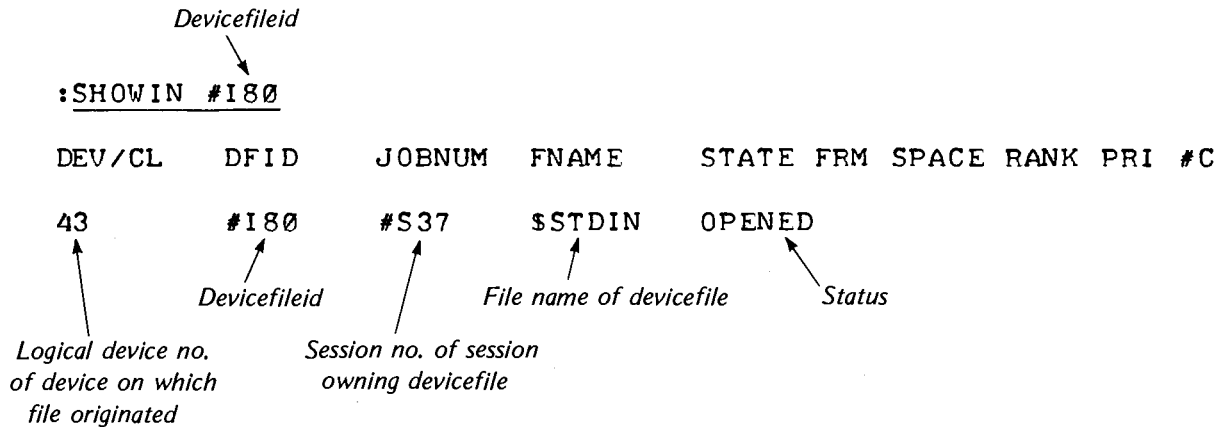


Figure 8-1. Input/Output Spooling

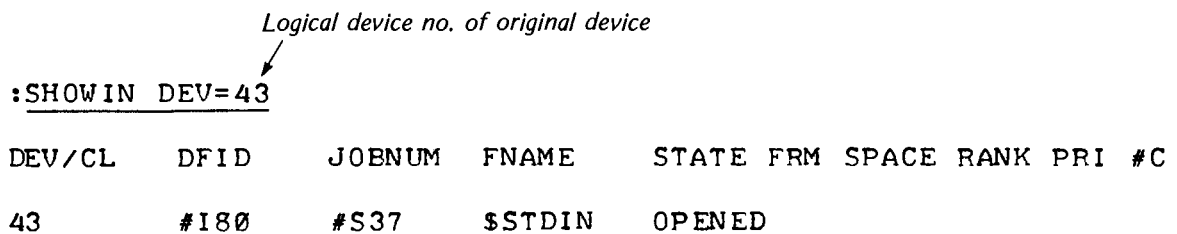
OBTAINING INFORMATION ABOUT INPUT DEVICEFILES. You can obtain summarized data about all input devicefiles existing in the system, or specific information about all or any of them, by entering the `:SHOWIN` command. As an example, suppose you want to determine how many devicefiles currently exist, how many of these are spooled devicefiles, and what states they are in. You can display this information by issuing the `:SHOWIN` command, as follows:



With the `:SHOWIN` command, you can also list information about individual input devicefiles. For instance, suppose that you wanted to determine the status of a devicefile identified by the identifier `#I80`. You would enter:



If you do not know the devicefileid of the devicefile whose status you want to determine, you may request the status display by entering either the logical device number or device class name of the device on which the file originated:



If you want to list the status of all devicefiles currently being used by your session/job, enter simply:

Omit parameter list

:SHOWIN

DEV/CL	DFID	JOBNUM	FNAME	STATE	FRM	SPACE	RANK	PRI	#C
43	#I80	#S37	\$STDIN	OPENED					

Suppose you wish to report information about the devicefiles being used by all sessions (but not jobs) in the system. You would enter:

*Request for data on devicefiles
for all sessions*

:SHOWIN JOB=@S

DEV/CL	DFID	JOBNUM	FNAME	STATE	FRM	SPACE	RANK	PRI	#C
20	#I81	#S38	\$STDIN	OPENED					
26	#I36	#S18	\$STDIN	OPENED					
34	#I58	#S26	\$STDIN	OPENED					
42	#I64	#S28	\$STDIN	OPENED					
43	#I80	#S37	\$STDIN	OPENED					
51	#I35	#S17	\$STDIN	OPENED					

6 FILES (DISPLAYED):

0 ACTIVE
0 READY; INCL 0 SPOOFLES, 0 DEFERRED
6 OPENED; INCL 0 SPOOFLES
0 LOCKED; INCL 0 SPOOFLES
0 SPOOFLES: 0 SECTORS

You can request information according to devicefile state. For instance, if you wanted to list all OPENED devicefiles being used by your current session/job, you would enter:

*Request for data on all OPENED devicefiles
used by this session/job*

:SHOWIN OPENED

DEV/CL	DFID	JOBNUM	FNAME	STATE	FRM	SPACE	RANK	PRI	#C
43	#I80	#S37	\$STDIN	OPENED					

You can also request displays of devicefile information by various combinations of qualifications (devices, sessions/jobs, and states). As an example, suppose you wanted to display information about all OPENED input devicefiles used by all sessions (but not jobs) in the system. You would enter:

*Requests data on OPENED devicefiles
used by all sessions*

:SHOWIN JOB=@S; OPENED

DEV/CL	DFID	JOBNUM	FNAME	STATE	FRM	SPACE	RANK	PRI	#C
20	#181	#S38	\$STDIN	OPENED					
26	#136	#S18	\$STDIN	OPENED					
32	#185	#S41	\$STDIN	OPENED					
34	#158	#S26	\$STDIN	OPENED					
42	#164	#S28	\$STDIN	OPENED					
43	#180	#S37	\$STDIN	OPENED					
50	#184	#S40	\$STDIN	OPENED					
51	#135	#S17	\$STDIN	OPENED					

8 FILES (DISPLAYED):
0 SPOOFLES: 0 SECTORS

Many other combinations are possible; see the Reference Specifications for the :SHOWIN command.

OUTPUT DEVICEFILES

Output devicefiles are files composed of data produced by user programs destined for output devices such as line printers, card punches, magnetic tape units, and graphic plotters. There are two primary types of output devicefiles:

- Session/job listing output devicefiles.
- Other output devicefiles.

SESSION/JOB LISTING DEVICEFILES. Each session/job-accepting device is assigned a corresponding default output device during system configuration. The default device may be assigned by logical device number or device class name. For each session/job submitted on a given accepting input device, the corresponding default output device becomes the destination of the session/job listing devicefile. (For batch jobs, you can override this default device by specifying another device in the OUTCLASS= parameter of the :JOB command.) There is exactly one session/job listing devicefile assigned to each executing session/job. This file always assumes the system-defined file name \$STDLIST. This devicefile is de-allocated at session/job termination.

OTHER OUTPUT DEVICEFILES. A session/job may create other output devicefiles by specifying a logical device different from the session/job listing device. If this device is a magnetic tape unit, the Computer Operator must allocate an available device. Other devices are allocated by MPE.

ACCESSING DEVICEFILES. Your program accesses an output devicefile by opening that file. If the file is a magnetic tape unit, MPE requests the Console Operator to assign it as follows:

1. If you have requested the tape unit by device class name, the Operator can specify any free device with this class name for the file.
2. If you have requested a specific tape unit by its unique logical device number, the Operator must assign this particular unit if it is available.

If the device requested is not a tape unit, MPE automatically grants access if the device is available. A non-sharable device is considered to be available if it is unowned, spooled, or owned by the requesting session/job and requested by logical device number.

When a session/job opens an output devicefile, special forms may be requested in the file-opening request. As a result, a user-forms message appears on the Operator's console along with a request to mount the forms:

1. If you have requested a device by device class name, the Operator is asked to mount the forms on any free device in the class.
2. If you have requested a specific device (by its logical device number), the Operator is asked to mount the forms on the device requested if it is available.

When you have requested special forms on a line printer, MPE initiates a dialogue with the Operator to direct him to align the forms. A standard record of this format appears on the printer, followed by a Console message which asks the Operator if the forms are aligned properly:

```

                                Column
                                133
                                ↓
    0.....:.....1.....:.....2.....:.....3...  ...  ...3..
  
```

This transaction is repeated until the Operator indicates that proper alignment is achieved. Assuming proper alignment instructions, the file can then be output beginning at the proper, user-defined position.

If special forms are already mounted on a device, and a devicefile not requiring special forms is assigned to this device, MPE automatically asks the Operator to mount standard forms.

If an output devicefile is directed to a line printer, MPE automatically prints a header page identifying the job that produced the file. When all accessing programs close the file, a trailer page is also printed. If an output devicefile is directed to a card punch, MPE automatically punches header and trailer cards identifying the job that produced the file.

NOTE

Output of these header and trailer records can be enabled or disabled by the Console Operator commands =HEADON and =HEADOFF, respectively.

OUTPUT SPOOLING. When a non-sharable output device is spooled, the device (called the spooler) is acquired by (belongs to) the spooling program. This program (process) is activated when the spooling operation is enabled. A user program attempting to write data to a spooled output device writes the outbound records to an OPENED spooled devicefile instead of the device itself. Each time the user program writes a record destined for the output device, the record is sent to the OPENED spooled devicefile. When the user program closes the output file, this spooled devicefile achieves the READY state. (In some cases, when the READY devicefile is being exclusively accessed by MPE with other accesses prohibited, the devicefile enters the LOCKED state.) The output spooler program constantly scans for spooled output devicefiles that have achieved the READY state. If the output device (spooler) is available, the spooler program selects a READY file for processing. This action changes the READY file to ACTIVE status, and the output to the device (spooler) commences. The normal transition for spooled output devicefiles is:

1. OPENED: Being accessed by a user program or MPE (being written upon).
2. READY: Available and waiting ACTIVE status or deletion by the Operator.
3. LOCKED: Being accessed exclusively by MPE, with other accesses prohibited.
4. ACTIVE: Currently being output to the destination device.

This transition is summarized in figure 8-1.

To summarize: the operation of the output spooler is to continually check for the presence of READY output spooled devicefiles. When the spooler finds a READY file and the output device is available, the file state changes from READY to ACTIVE and output to the device from this file commences. READY files are selected on the basis of their output priority and by age among those of equal priority.

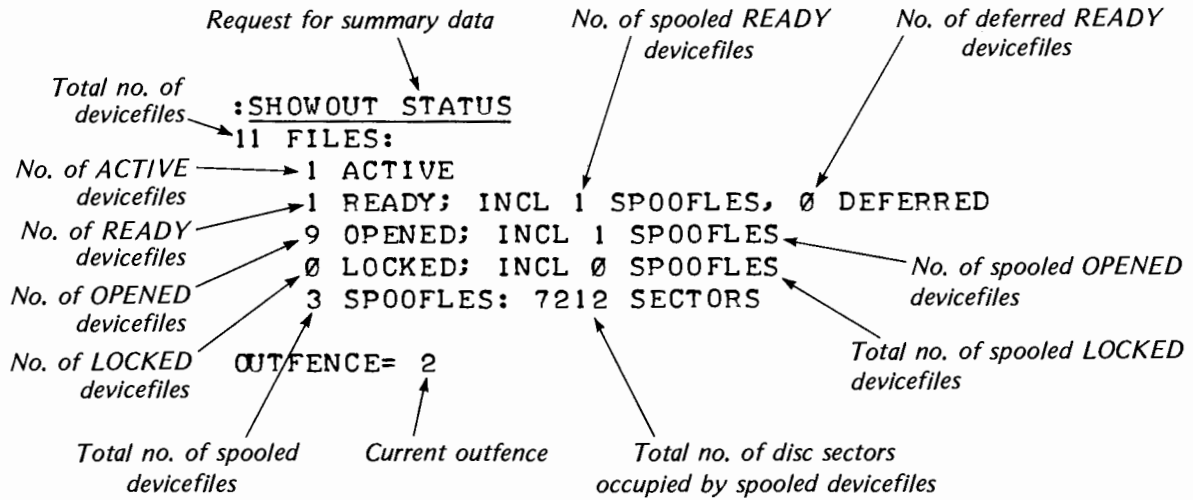
The Console Operator can set an *oudfence*, as explained in Section VI. Output devicefiles with a priority less than or equal to the current *oudfence* are deferred; such devicefiles are not selected for spooling by output spoolers. (Since the *oudfence* cannot be set to a value less than one, devicefiles with output priorities of one are not output.) An output spooler which fails to find a READY spooled devicefile for copying to a device enters a WAITING state. It is automatically re-activated when a spooled devicefile becomes READY for copying.

The Console Operator can also change the output priority, destination, and number of copies requested for an OPENED or READY spooled opened devicefile.

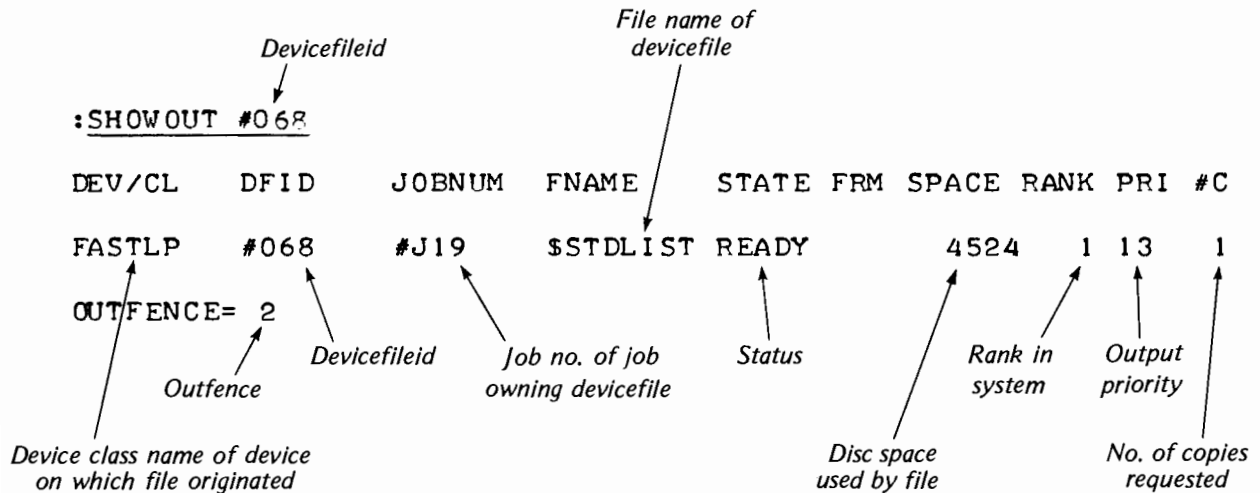
To minimize disc occupancy, extents of output spoolfiles are released as the final (possibly the only) copy is being made. An interruption of the last copy (caused, for instance, by system failure or Operator intervention) causes the remaining extents to be copied when the spoolfile is re-scheduled. In such cases, there is always some overlap with the previous output, so that no data is lost. Interruption of other than the last copy, however, causes the entire copy to be output from the beginning upon re-scheduling.

When a spoolfile with special forms currently on the output device is interrupted (by Operator intervention or system failure), an indication that special forms are required is again issued to the Console Operator when the spoolfile is again selected for spooling. In these cases, there is always sufficient overlap with the previous output so that alignment of the forms can be re-established visually and manually, using the overlapped lines.


OBTAINING INFORMATION ABOUT OUTPUT DEVICEFILES. As with input devicefiles, you can obtain summarized data about all output devicefiles existing in the system or specific information about any or all of them. This is done by entering the :SHOWOUT command. For instance, to display the total number of output devicefiles currently existing, the number of those that are spooled, and the states that they are in, enter:



To list information about an individual output devicefile, you can reference its devicefile identifier in the :SHOWOUT command:



You can also request the status of a devicefile by using the logical device number or device class name of the device for which the file is destined in the :SHOWOUT command:


Logical device no. of device


```
:SHOWOUT DEV=43
```

DEV/CL	DFID	JOBNUM	FNAME	STATE	FRM	SPACE	RANK	PRI	#C
43	#089	#S37	\$STDLIST	OPENED					

OUTFENCE= 2

To display the status of all output devicefiles currently used by your session/job, enter:


Omit parameter list


```
:SHOWOUT
```

DEV/CL	DFID	JOBNUM	FNAME	STATE	FRM	SPACE	RANK	PRI	#C
43	#089	#S37	\$STDLIST	OPENED					

OUTFENCE= 2

You can list information about the devicefiles being used by all jobs (but not sessions) or all sessions (but not jobs). For example, to list information about all job devicefiles, enter:

*Request for data on devicefiles
for all jobs*


```
:SHOWOUT JOB=@J
```

DEV/CL	DFID	JOBNUM	FNAME	STATE	FRM	SPACE	RANK	PRI	#C
FASTLP	#068	#J19	\$STDLIST	READY		4524	1	13	1
FASTLP	#094	#J21	\$STDLIST	OPENED		384		13	1
4	#084	#J22	\$STDLIST	ACTIVE		2304	1	13	1

3 FILES (DISPLAYED):
 1 ACTIVE
 1 READY; INCL 1 SPOOFLES, 0 DEFERRED
 1 OPENED; INCL 1 SPOOFLES
 0 LOCKED; INCL 0 SPOOFLES
 3 SPOOFLES: 7212 SECTORS

OUTFENCE= 2

You can request information according to devicefile state. For example, to list all OPENED output devicefiles being used by your session/job, enter:

*Request for report on all OPENED devicefiles
used by this session/job*

:SHOWOUT OPENED

DEV/CL	DFID	JOBNUM	FNAME	STATE	FRM	SPACE	RANK	PRI	#C
43	#089	#S37	\$STDLIST	OPENED					

OUTFENCE= 2

You can request displays of output devicefile information by various combinations of qualifications (devices, sessions/jobs, and states). As an example, suppose you wanted to list information about all OPENED output devicefiles used by all sessions in the system. You would enter:

*Request for data on OPENED output devicefiles
used by all sessions*

:SHOWOUT JOB=@S; OPENED

DEV/CL	DFID	JOBNUM	FNAME	STATE	FRM	SPACE	RANK	PRI	#C
20	#090	#S38	\$STDLIST	OPENED					
26	#037	#S18	\$STDLIST	OPENED					
27	#095	#S42	\$STDLIST	OPENED					
32	#093	#S41	\$STDLIST	OPENED					
34	#064	#S26	\$STDLIST	OPENED					
42	#073	#S28	\$STDLIST	OPENED					
43	#089	#S37	\$STDLIST	OPENED					
50	#092	#S40	\$STDLIST	OPENED					
51	#036	#S17	\$STDLIST	OPENED					

9 FILES (DISPLAYED):

0 SPOOFLES: 0 SECTORS

OUTFENCE= 2

REQUESTING UTILITY OPERATIONS

SECTION

IX

MPE allows you to request various utility operations. Some of these are general functions that you may use at any time, while others are intended primarily for special applications. These operations include:

- Communicating with other users or the Console Operator (Page 9-1).
- Changing the input or output speed of your terminal, on-line (Page 9-5).
- Reading paper tapes not using the standard X-OFF delimiter (Page 9-6).
- Acquiring Resource Identification Numbers (RINs) for resource management (Page 9-10).
- Displaying the current date and time, as recorded by the System Clock (Page 9-12).

These operations and related background information are discussed in this section, with cross-references to appropriate command specifications in Section II.

COMMUNICATION BETWEEN USERS

In a multi-user environment, users frequently need to communicate with one another and with the Console Operator. MPE allows this type of communication through various user and operator commands, as illustrated in figure 9-1.

USER/TO-USER COMMUNICATION

As a system user, you may need to transmit messages to other users. For instance, if you are co-operating with another user in the use of some program or subroutine, you might want to inform him of updates to that software. To do this, you request a list of all users currently logged on, by entering the :SHOWJOB command (discussed on page 5-1); then, if the user is running a session/job, you can transmit your message to him by entering the :TELL command, described below. As another example, if you are working at a remote site, you might wish to request assistance with a peripheral device located closer to the computer. In this case, you can enter :SHOWJOB to obtain a list of everyone logged on, select an appropriate user located near the device, and enter a :TELL command directing your request to him.

You can issue the :TELL command in this way:

```
:TELL JONES.ACT1; USE PROGB, NOT PROGA.
```

Receiver's session/job identity *Message text*

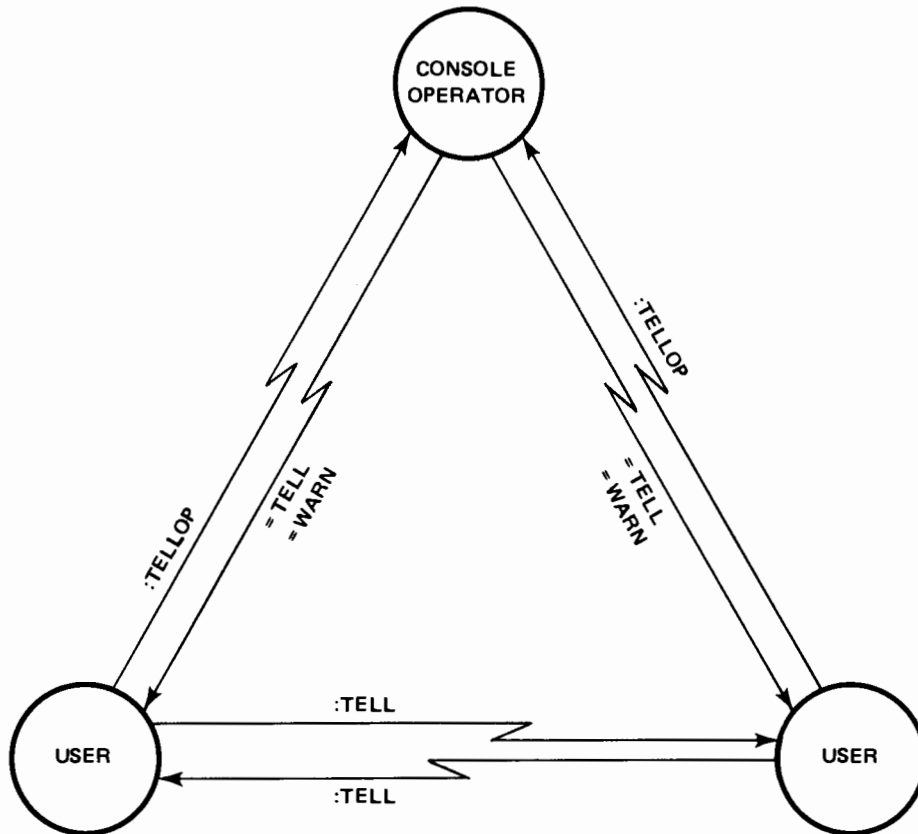


Figure 9-1. User/Operator Intercommunication

This message appears on the receiving user's standard list device like this:

```
FROM SMITH.ACT1 / USE PROGB, NOT PROGA.
```

Sender's session/job identity Message text

Notice that, in this example, the message was transmitted by referencing the receiver's session/job identity in the :TELL command. But what if several users were logged onto the system under the same identity? In that case, MPE arbitrarily selects only one of them, and transmits the message to him (with NO error message or other warning to you.) So, to ensure that you get the right user, you may reference that user's *session/job number* rather than his identity in the :TELL command, as follows:

```
:TELL #S16; USE PROGB, NOT PROGA.
```

Receiver's session number Message text

The session/job number is unique to each session/job within MPE, and positively identifies the recipient you wish.

NOTE

You can obtain both the session/job identity and session/job number for any user logged on by entering the :SHOWJOB command.

Normally, when you send a message, it is queued for output as high as possible among the receiving user's input/output requests (although it does not interrupt any reading or writing currently in progress). If, however, the intended receiver cannot actually receive your message, you are informed of this by an error message on your standard list device.

The message text transmitted must not exceed 66 characters, minus the length of your (sender's) session/job identity. If it does, MPE rejects it.

If the receiving user does not want to accept any messages from other users, he may have placed his terminal in the *quiet mode*, described below. In this case, MPE rejects your message and informs you with:

```
USER NOT ACCEPTING MESSAGES
```

At log-on time, every session/job's standard list device is enabled to accept messages of all kinds. But, should you wish to prevent messages from other users or non-critical messages from the Console Operator from interrupting your session/job, you can place your standard list device in the *quiet mode* by entering:

```
:SETMSG OFF
```

Notice that although *non-critical* messages from the Console Operator (transmitted via the =TELL operator command) are suppressed by *quiet mode*, *critical* messages from the Operator (issued via the =WARN command) are not — they override quiet mode in all cases.

Quiet mode remains in effect until your session/job terminates or until you revoke it by entering

```
:SETMSG ON
```

USER-TO-OPERATOR COMMUNICATION

On occasion you may wish to send information or transmit requests to the Console Operator. For instance, you may ask him to perform functions such as mounting a magnetic tape, aligning forms on a printer, or simply supplying certain information that you need. You can send such messages by entering the :TELLOP command, as in the following example.

```
:TELLOP WHAT TIME WILL SHOP CLOSE TONIGHT?
```

The message text appears on the operator's console, preceded by the time it was transmitted and your session/job number. Like messages transmitted between users, this message is printed as soon as possible without interrupting any console input/output currently in progress.

The message text transmitted must not exceed 56 characters. If it does, MPE rejects it.

OPERATOR-TO-USER COMMUNICATION

As a system user, you may receive various messages from the Console Operator. They may be *non-critical* messages that require no immediate action on your part. Such messages appear in this format:

```
FROM/OPERATOR/ SHOP CLOSES AT 4:30 PM TODAY
```

These messages, which the operator issues with the =TELL command, are queued as high as possible among your input/output requests but do not interrupt any input/output currently in progress.

You may also receive *critical* messages from the operator. These typically announce emergency conditions or ask you to immediately perform some task. They appear in the following format:

```
WARN/ LOG-OFF ASAP.  SYSDUMP WILL BEGIN IN 2 MINUTES.
```

Critical messages, which the operator issues with the =WARN command, appear on your standard list device immediately upon receipt; they automatically interrupt any input/output currently in progress.

(The operator commands =TELL and =WARN are described in *Console Operator's Guide*.)

CHANGING TERMINAL SPEED

MPE supports terminals that run at speeds ranging from 10 to 240 characters per second (cps). Some of these terminals offer adjustable input/output speeds. Prior to logging on with such a terminal, you can select the input/output speed you desire simply by adjusting a switch located on the terminal; MPE automatically senses the speed selected. After you have logged on, you can still alter the input or output speed without logging off. This feature is useful if you wish to dynamically alter input/output speed during a session. Suppose, for instance, that you log-on at a CRT terminal with input/output speed set at 240 cps. Later in your session, you encounter lengthy output displays that roll off the top of your screen before you have time to read them. To preclude this, you decide to slow the terminal speed to 10 cps. You do this by performing the following steps:

1. Enter the MPE `:SPEED` command, as in the following example:

```
      Input speed      Output speed
      |                |
      v                v
: SPEED 10, 10
```



MPE responds by printing the following message, at the old output speed:

```
CHANGE SPEED AND INPUT "MPE":
```

2. Manually change the speed control switches on the terminal to the desired settings. (In this case, 10 cps for both input and output.)
3. Enter the characters "MPE" to verify the new input speed. If it is verified, you will be prompted for another command; continue with your session. If it cannot be verified, this message is transmitted to the terminal:

```
SPEED NOT VERIFIED - OLD SPEED RETAINED
```

NOTE

If you have manually set the terminal to a new speed (Step 2), this message does *not* appear in decipherable form. Re-set the terminal to the old speed and return to Step 1.

If you did not manually change the speed of the terminal (Step 2), the message will appear in readable form. Return directly to Step 1.

MPE accepts the `:SPEED` command during a break as well as during session mode.

If you use the `:SPEED` command for a fixed-speed terminal or enter an invalid input or output speed, MPE issues an error message such as:

```
ILLEGAL DEVICE OR INVALID OUTPUT SPEED.
```

You can also change the terminal speed programmatically, by using the `FCONTROL` intrinsic discussed in *MPE Intrinsic Reference Manual*.

USING PAPER TAPES AT THE TERMINAL

Terminal records produced by MPE are always delimited by a carriage-return character, followed by a line-feed character, followed by an X-OFF control character. Output of the X-OFF character causes no action at the terminal itself; its sole purpose is to provide a special tape-reader directive to be punched on tape if the terminal in use supports a paper tape reader/punch unit. This X-OFF directive functions as part of a hand-shaking arrangement between the terminal subsystem and MPE, as shown in figure 9-2. The X-OFF character turns the tape unit's reading mechanism off whenever it is encountered, usually at the end of a tape record. Prior to the next read, MPE transmits an X-ON character to re-activate the tape unit. The most common terminal/tape-unit combination used with MPE is the HP 30124A Teleprinter Terminal (ASR 33 or 35), usually referred to as a "Teletype."

When you are running a program that reads a tape from an HP 30124A Terminal and that tape was produced by MPE on the same type of terminal, or was manually punched by an operator who included X-OFF characters, the tape is read in the normal way with no special action required. But, if you are working at an HP 30124A or some other terminal that includes a tape reader but you want to read a paper tape prepared without the proper inclusion of X-OFF characters, you must use the :PTAPE command. This command transfers the tape input to an ASCII file on disc, which you name as a command parameter. During this operation, MPE first reads the input from the tape into a 32,767-byte data segment (buffer), as illustrated in figure 9-3. When you enter a CONTROL-Y (Y^c) character (indicating the end of the data to be read), or when this character is encountered on the tape, MPE copies the complete buffer contents into the named disc file and returns control to the keyboard. The information from the tape is now available in the disc file, and can be read by any program. In the disc file, each record appears in the same order as on the tape file.

NOTE

When the tape is read, MPE deletes all carriage-return and line-feed characters (plus any X-OFF characters present) from the data stream. But, although it is completely ignored by MPE, the X-OFF character is not ignored by terminals that use that character. Therefore, if the :PTAPE command is used in conjunction with an HP 30124A to read a tape containing X-OFF characters, the terminal must be manually re-started after each X-OFF is encountered.

To illustrate how to read a tape that does not contain X-OFF characters using an HP 30124A Terminal, consider such a tape on which a four-record FORTRAN program is stored. This program is to be compiled, prepared, and executed by MPE. You could proceed as follows:

1. Load the paper tape onto the terminal/tape unit.
2. Create an appropriate disc file, into which the input will be read, by entering the :BUILD command. This file should be an ASCII file, typically specified for variable-length records.

```
:BUILD JUNKFILE;REC=126,1,V,ASCII
```

Be certain that the record length you request (with the REC= parameter) is at least as great as the longest record to be read from the tape.

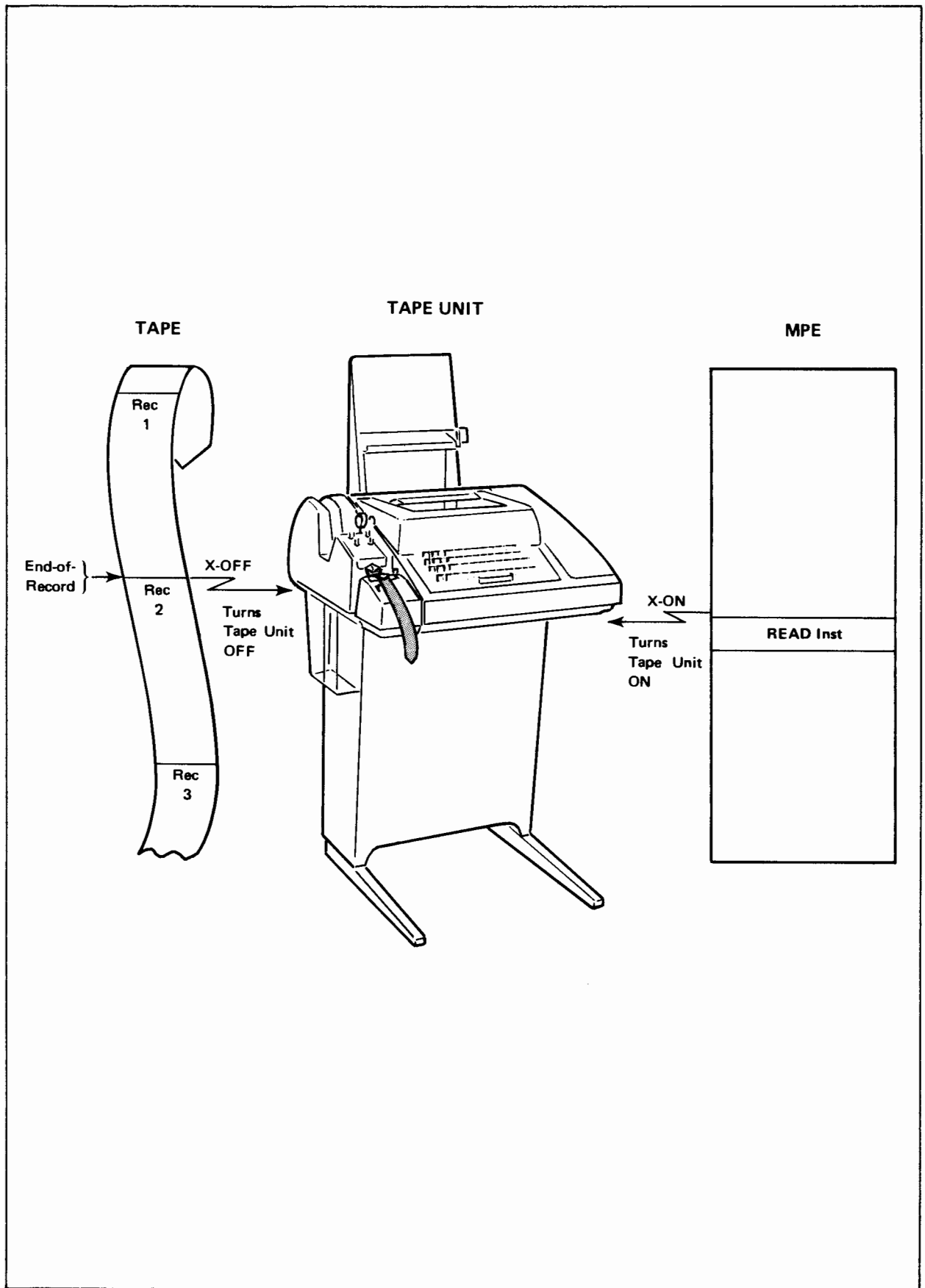


Figure 9-2. X-OFF/X-ON Hand-shaking Arrangement

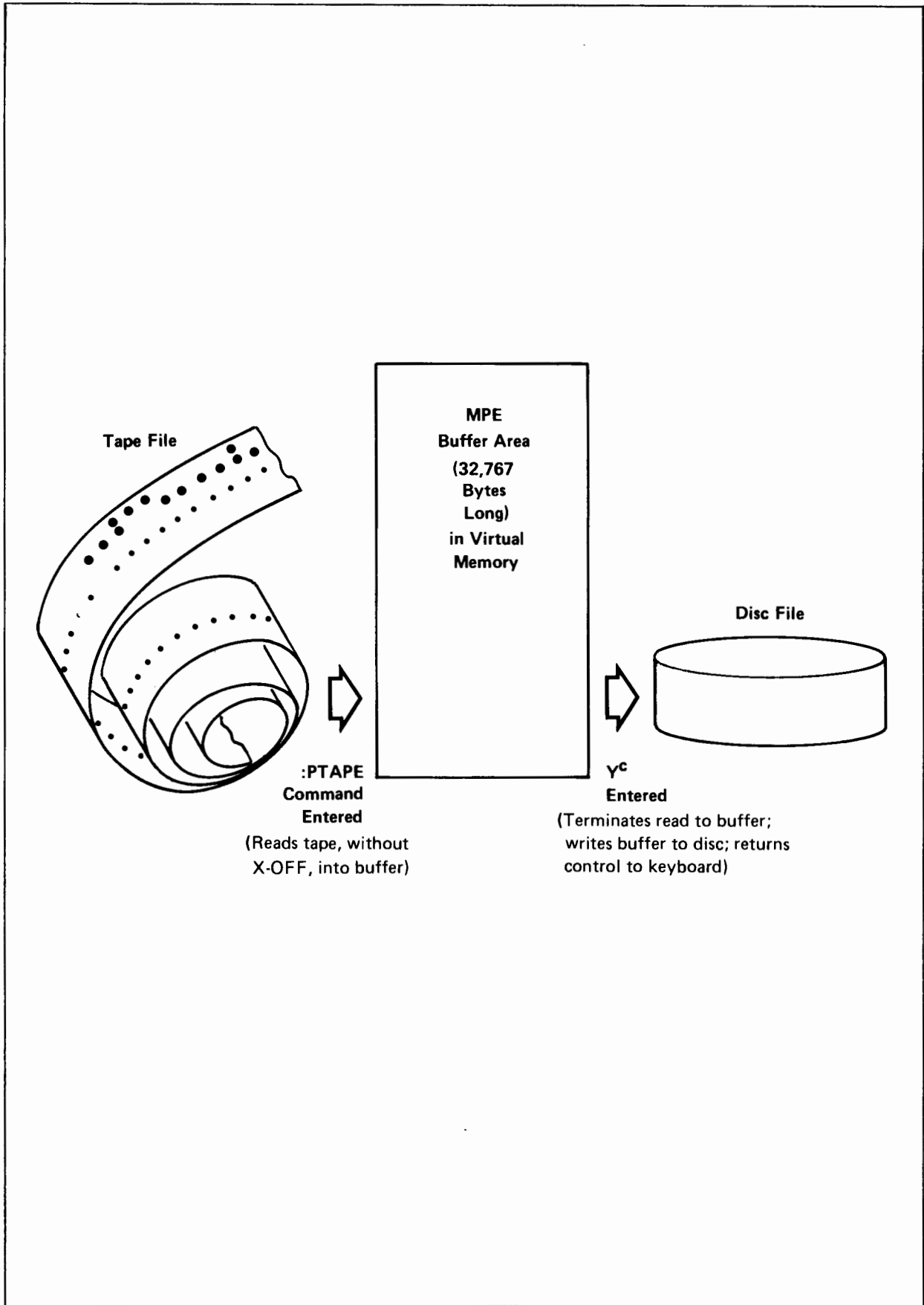


Figure 9-3. :PTAPE Command Operation

3. Enter the :PTAPE command to copy the tape file onto disc.

```
:PTAPE JUNKFILE
```

4. Manually start the tape reader. MPE copies the tape, echoing its records at the terminal, as shown below. Copying continues until MPE encounters Y^c on the tape or until you enter this character from the keyboard. If the tape runs through to the end, then shut the reader off and enter Y^c at this time.

```
$CONTROL FREE
PROGRAM DUMMY
DISPLAY "WRITE THIS"
999 END
```

NOTE

Records containing special control characters such as Hc or Xc are honored, even though these characters are echoed on the teletype.

5. Terminal activity pauses while MPE unpacks the data from the buffer. When that operation is complete, MPE displays a colon as a prompt for your next MPE command. At this point, you could enter a command that compiles, prepares, and runs the program just stored on disc, as shown below:

```
→ :FORTGO JUNKFILE
```

```
PAGE 0001 HP32102B.00.0
```

```
$CONTROL FREE
PROGRAM DUMMY
DISPLAY "WRITE THIS"
999 END
```

```
**** GLOBAL STATISTICS ****
**** NO ERRORS, NO WARNINGS ****
TOTAL COMPILATION TIME 0:00:01
TOTAL ELAPSED TIME 0:00:30
```

```
END OF COMPILE
```

```
END OF PREPARE
```

```
WRITE THIS
END OF PROGRAM
```

NOTE

If you are attempting to use :PTAPE with a terminal using a tape cassette, or with a device other than a terminal, you should do so with caution. Primarily, you should be sure that the input to be read from the tape does not exceed the 32,767-byte buffer size noted above — if it does, reading halts when the buffer is full (prior to the end of data on the tape), the buffer contents are transferred to the disc file at this time, and the operation is terminated prematurely.

In addition, you should be certain that the length specified for the disc file records is sufficient to allow for the longest record on the tape. Otherwise, the MPE File Management System detects an error when it attempts to copy the data from the buffer into the file named in the :PTAPE command, and aborts that operation with an error message, indicating File System Error 43. (See *MPE Intrinsic Reference Manual* for a description of this error.)

MPE also handles paper tape input/output via two other standard subsystems:

- HP 30104A Punched Tape Reader Subsystem
- HP 30105A Tape Punch Subsystem

Input/output for these subsystems requires no special MPE commands, and is discussed in the reference manuals covering the subsystems.

MANAGING RESOURCES THROUGH RESOURCE IDENTIFICATION NUMBERS (RINS)

MPE permits its users to manage a particular hardware or software resource shared by a set of sessions or jobs, so that no two of these sessions or jobs can use the resource at the same time. This type of resource management is carried out through the use of global Resource Identification Numbers (RINs). Because these RINs are manipulated entirely through intrinsics, the techniques for using RINs and their various applications are described in *MPE Intrinsic Reference Manual*. Before any users can use a global RIN, however, one user must acquire it from the system by entering the :GETRIN command; when all users are finished with the RIN, the user who acquired it returns it to the system by entering the :FREERIN command. These operations and commands are discussed below.

ACQUIRING GLOBAL RINS

Before you and any other users cooperating in the use of a particular global RIN can use it in your sessions and jobs, you must first acquire it from the MPE RIN pool. You do this by entering the :GETRIN command, typically during a session. With this command, you must also assign an arbitrary

password for the RIN that aids in restricting its use to proper users only. MPE responds by printing the RIN on the line following the command:

```
:GETRIN OURPASS ←—— RIN password  
RIN: 2 ←—— RIN
```

You give this RIN and the associated password to all other cooperating users, so that all of you can lock and unlock it. (The techniques for locking and unlocking the RIN, and passing it and its password as intrinsic parameters, are described in *MPE Intrinsic Reference Manual*.) All cooperating users can continue using the RIN for as long as they desire — days or weeks if necessary — until it is released, as described below. If, when you enter the :GETRIN command, all RINs currently available in the system are acquired by other users, MPE rejects the request and issues the message:

RIN TABLE FULL

In this case, you must wait until one of the RINs becomes available or see your System Manager about raising the maximum number of RINs that can be assigned.

FREEING GLOBAL RINS

When all cooperating users have finished using a particular global RIN in their sessions and jobs, the *owner* of the RIN can de-allocate it, returning it to the MPE RIN pool.

NOTE

MPE regards the user who originally acquired the RIN (by issuing :GETRIN) as the *owner* of the RIN. This means that *only that user* may release the RIN. (RIN ownership is permanently recorded in the system, and is reflected on all back-up tapes created by the System Manager or Supervisor.)

To release a RIN, simply enter the :FREERIN command followed by the RIN itself. For instance,

```
:FREERIN 2
```

If MPE releases the RIN successfully, this is indicated by a prompt for a new MPE command. If, however, the RIN cannot be released because it is being used by another user, MPE rejects the request and issues the following message:

RIN CURRENTLY IN USE

If you are not the owner of the RIN you are attempting to release, MPE rejects the request and issues this message:

RIN NOT ALLOCATED TO THIS USER

DISPLAYING CURRENT DATE AND TIME

You can display the current date and time, as recorded by the system clock, by entering the `:SHOWTIME` command. For example:

```
:SHOWTIME  
MON, AUG 11, 1975, 9:33 AM
```

When using MPE at any terminal or batch input device, you may encounter the following types of messages, all discussed in this section:

Command Interpreter Error Messages, reporting fatal errors that occur during the interpretation or execution of an MPE command.

Command Interpreter Warning Messages, reporting unusual conditions that occur during command interpretation or execution but that may not necessarily affect the processing of your session or job.

System Messages, that denote miscellaneous conditions that terminate or otherwise affect your session/job, such as an abort requested by the System Supervisor or Console Operator.

Operator Messages, which are messages sent to you by the Console Operator.

User Messages, which are messages sent to you by other users currently running sessions or jobs.

In addition, you may also encounter the following type of message, described in *MPE Intrinsic Reference Manual*:

Run-Time Messages, denoting conditions that abort your running program (provided that an appropriate error trap has not been armed).

When using the MPE Segmenter Subsystem, you may receive the following type of message, described in *MPE Segmenter Reference Manual*:

Segmenter Error Messages, reporting errors detected by the Segmenter.

Other messages may be received only at the System Console. These messages, which are described in *MPE Console Operator's Guide*, are:

Console Messages, including:

- Status Messages that indicate the current status of sessions/jobs or input/output devices.
- Input/Output Messages that request service for, and report errors on, input/output devices.
- User Messages, sent by users to the Console Operator.

System Error/Failure Messages.

COMMAND INTERPRETER ERROR MESSAGES

When MPE detects an error during interpretation or execution of an MPE command, it suppresses execution of that command and displays an error indication on the session/job listing device in the format shown below:

`ERR errnum [,detail] [message]`

errnum A number ranging from 0 to 951, identifying one of the errors described in table 10-1. These error numbers are grouped as follows:

- 0 - 19: General errors not related to specific command parameters.
- 20 - 47: Errors relating to command parameter syntax.
- 48 - 99: Errors relating to specific commands.
- 100 - 199: Errors encountered by the File Management System, or within the file directory.
- 200 - 249: Errors encountered by the CREATE intrinsic (that creates a process) or by the MPE Loader.
- 250: Error encountered by the MPE Segmenter. (See *MPE Segmenter Reference Manual*.)
- 900 - 951 Errors on \$STDIN or \$STDLIST file.

detail A number that (unless otherwise stated in table 10-1) refers to the erroneous parameter element in the command. When counting parameter elements, count from the left, beginning with the element immediately following the command name as Element No. 1; consider every delimiter, including equal signs, and omitted items.

message A message describing the error, also shown in table 10-1. In sessions, this message is optional and may be requested by entering any character other than a carriage return following:

`ERR errnum [,detail]`

In batch jobs, the message appears automatically. When an error occurs while you are trying to initiate a job or session, the error number is printed but no *message* appears (jobs) or can be requested (sessions).

When the error occurs during an interactive session, MPE suppresses execution of the erroneous command, displays the error indication (*errnum* parameter, sometimes followed by *detail* parameter) showing the type of error, and returns control to your terminal. If you desire an explanation of the error, you can request it by entering any character other than a carriage-return immediately after the error indication. In response, MPE displays the error message (*message* parameter) on the next line. (For further discussion of this message, see table 10-1.) If you do not require an explanation of this error, enter a carriage-return directly after the error indication. MPE then prints a colon, prompting you for another MPE command. (See examples in Section III.)

When the error occurs during a batch job (and you have not preceded the erroneous command by a :CONTINUE command as discussed in Section IV), MPE suppresses execution of the erroneous command, prints the entire error description (*errnum*, *detail*, and *message* parameters) on your standard listing device, ignores all subsequent commands in the job, and aborts the job. (See Section IV for examples.)



Table 10-1. Command Interpreter Error Messages

ERRNUM	MESSAGE
0	<p>JOB NOT YET DEFINED</p> <p>The first command entered was not :JOB, :HELLO, or :DATA, required to initialize an input stream. Enter such a command.</p>
1	<p>UNKNOWN COMMAND</p> <p>The command entered was not recognized as a legal MPE command. Enter a correct command.</p>
2	<p>ABNORMAL PROGRAM TERMINATION</p> <p>Your program terminated with an error condition (the high-order bit of the job control word set). (Calling the QUIT or QUITPROG intrinsic, setting Bit 0 of the job control word to 1, issuing an :ABORT command during a break, or killing a process does this; HP 3000 subsystems set this state when detecting serious errors.) Debug and re-run program.</p>
3	<p>WRONG (JOB/SESSION) MODE</p> <p>This command is not permitted in the defined job or session mode. Enter a permitted command.</p>
4	<p>INSUFFICIENT CAPABILITY</p> <p>You do not have the capability required to execute this command or one of its particular options. Check your capability against command requirements.</p>
5	<p>TOO MANY PARAMETERS</p> <p>The command image contained too many parameters or exceeded 255 characters. Check the parameter list and re-enter the command correctly.</p>
6	<p>INSUFFICIENT PARAMETERS</p> <p>Required parameters were omitted from the command parameter list. Check the parameter list and re-enter the command correctly.</p>
7	<p>MISSING COLON</p> <p>This command, encountered in a batch job, did not contain a colon as the first character of the command image. (Continuation lines must also have such colons.) Correct the command and re-run the job.</p>

Table 10-1. Command Interpreter Error Messages (Continued)

ERRNUM	MESSAGE
8	<p>ILLEGITIMATE ACCESS</p> <p>Access to MPE was denied because one of the following conditions occurred:</p> <ol style="list-style-type: none"> 1. You did not specify a log-on group and also had no home group. Specify a log-on group when re-entering the log-on command. 2. The account, user, name, specific group, or home group specified does not exist. See system manager, supervisor, or operator. 3. A password is required by the account, user, or group (other than home group) name, and you either supplied no password or supplied an incorrect one. Check to ensure that the correct password is supplied; if this fails, contact the system manager or account manager.
9	<p>UNACCEPTABLE DEVICE</p> <p>A :HELLO or :JOB command was entered on a device not currently configured to accept jobs/sessions; a :DATA command was entered on a device that does not presently accept data; or :HELLO was entered on a non-interactive terminal. Initialize input stream on an appropriate device.</p>
10	<p>INSUFFICIENT RESOURCES</p> <p>The operation requested was rejected because sufficient system resources (data segments, code segment table entries, process control blocks, and so forth) were not available for it. Request operation when it is more likely that such resources are available, or see system supervisor.</p>
11	<p>COMMAND NOT YET IMPLEMENTED</p> <p>The command entered is not legal in this version of MPE.</p>
12	<p>NOT ALLOWED PROGRAMMATICALLY</p> <p>The command cannot be entered with the COMMAND intrinsic.</p>
13	<p>VDD FULL</p> <p>A :DATA command was entered, but the virtual device directory was full and could not accomodate more :DATA files. Re-enter command later, or see system supervisor.</p>
14	<p>JOB OVERLOAD</p> <p>A request for job scheduling was not accepted because the job master table is full. Request scheduling later, or see system supervisor.</p>
15	<p>SUBSYSTEM NOT FOUND</p> <p>An MPE subsystem was requested that does not exist in the permanent file directory. See system manager.</p>

Table 10-1. Command Interpreter Error Messages (Continued)

ERRNUM	MESSAGE
16	<p>DISC I/O ERROR</p> <p>A disc input/output error occurred during processing of a command. Depending on the nature of the error, the command may succeed when re-issued.</p>
20	<p>SYNTAX ERROR</p> <p>A delimiter in the parameter list is not permitted in the command, or is not permitted at the point where it was detected. When a qualifying number is shown (<i>detail</i>), the bad delimiter is adjacent to the indicated parameter element (usually following it). Re-enter the command correctly.</p>
21	<p>PARAMETER NOT OPTIONAL</p> <p>A parameter was omitted in a position where the parameter is required. Re-enter the command correctly.</p>
22	<p>ILLEGAL PARAMETER</p> <p>A parameter was not recognized as legal. Check the parameter list, and re-enter the command correctly.</p>
23	<p>PARAMETER OUT OF BOUNDS</p> <p>The value specified for the indicated parameter was not within the allowable range. Refer to the command description for the permissible range, and re-enter the command correctly.</p>
24	<p>ILLEGAL PARAMETER IN THIS CONTEXT</p> <p>An otherwise legitimate parameter is not permitted in the specified format of the command, or it conflicts with some previous specification in the command. Check the command format, and re-enter the command correctly.</p>
25	<p>DUPLICATE PARAMETER</p> <p>The command was rejected because the same parameter appeared twice in the parameter list; re-enter the command without the duplicate parameter.</p>
26	<p>ILLEGAL KEYWORD</p> <p>A keyword in a command was not recognized as legal. Check the keyword, and re-enter the command correctly.</p>
27	<p>DUPLICATE KEYWORD</p> <p>The command was rejected because the same keyword appeared twice in the parameter list; re-enter the command without the duplicate keyword.</p>
28	<p>ILLEGAL KEYWORD IN THIS CONTEXT</p> <p>An otherwise legitimate keyword is not permitted in the specified format of the command, or it conflicts with some previous specification in the command. Check the command format, and re-enter the command correctly.</p>

Table 10-1. Command Interpreter Error Messages (Continued)

ERRNUM	MESSAGE
29	<p>ILLEGAL NAME</p> <p>A syntactically-invalid name was included in the command; most names are limited to eight alphanumeric characters or less, beginning with a letter. Check the format of the name.</p>
30	<p>INVALID NUMBER</p> <p>A syntactically-invalid number appeared in the command. This could be caused by a non-numeric character (other than a leading "+", "=", or "%"); an "8" or "9" for an octal number; or a number that is too large. Check the parameter list, and re-enter the command correctly.</p>
48	<p>TERMINAL/TYPE INCOMPATIBILITY</p> <p>For a :HELLO or :JOB command, the <i>TERM=termtype</i> parameter is not legitimate for the terminal being used. Check the terminal type versus the <i>termtype</i> specified.</p>
49	<p>BAD OUTCLASS</p> <p>For a :JOB command, the <i>OUTCLASS=device</i> parameter is not a legitimate device specification, or the device (device class) does not support serial output. Check the device requested against the <i>device</i> parameter.</p>
50	<p>MINIMUM CAPABILITIES OMITTED</p> <p>One of the following violations occurred:</p> <ol style="list-style-type: none"> 1. SM capability was not included for the SYS account. 2. Neither IA nor BA capability was included. 3. A system manager or account manager attempted to cancel his own manager capability. <p>Check the caplist parameter, and re-enter the command properly.</p>
51	<p>FILESPEC LIMIT BELOW CURRENT COUNT</p> <p>A filespace limit was specified that was less than the current amount allocated. Re-specify the limit correctly.</p>
52	<p>CAPABILITY EXCEEDS ACCOUNT'S</p> <p>In an account manager command that creates or alters a group or user, the capability specified for group, user, or local attributes is not a subset of the account's capability or local attributes. Determine account's capability and respecify user's capability within these limits.</p>
53	<p>MAXPRI EXCEEDS ACCOUNT'S</p> <p>In an account manager command that creates or alters a user, specification of the user's maximum priority is higher than the account's maximum priority. Determine account's maximum priority, and respecify user's within these limits.</p>

Table 10-1. Command Interpreter Error Messages (Continued)

ERRNUM	MESSAGE
54	<p>LIMIT(S) EXCEEDS ACCOUNT'S</p> <p>Filespace, central processor time, and/or connect limit greater than the corresponding limits for the account were specified. Determine these values for the account, and re-specify user's limits accordingly.</p>
55	<p>ILLEGAL FILE NAME</p> <p>In a command reference for a file, an illegal file name appeared. The syntax for file names is:</p> <p style="padding-left: 40px;">filename [/lockword] [.gname[.aname]]</p> <p>Each name or lockword consists of eight characters or less, beginning with a letter. Check this syntax against the file name specified, and re-enter the command.</p>
56	<p>FILE EQUATION TABLE FULL</p> <p>The last :FILE command was rejected because no more space existed in the file equation table. The :RESET command can be used to provide space for additional entries by cancelling existing :FILE commands.</p>
57	<p>BACK FILE REFERENCE NOT FOUND</p> <p>A previous :FILE command for this session/job was referenced, in the <i>*formaldesignator</i> format, but could not be found. Check :FILE name.</p>
58	<p>TOO MANY BACK FILE REFERENCES</p> <p>A :FILE command had more than 155 following :FILE commands referring to it via the <i>*formaldesignator</i> back-file reference construct.</p>
59	<p>INVALID FILE DESIGNATOR</p> <p>The formal file designator in a :FILE or :RESET command was syntactically incorrect. Re-enter the command with correct syntax.</p>
61	<p>DEVICE OF WRONG TYPE</p> <p>A device of the wrong type was referenced in a command—for example, output was directed to a card reader or the :STORE command was applied to a non-tape file. Re-specify, with correct device.</p>
62	<p>DUPLICATE ACCESS SPECIFIED</p> <p>A <i>modelist:userlist</i> pair appears twice in the same :ALTSEC command (or system manager command) parameter list. Eliminate this redundancy and re-enter command.</p>
63	<p>ILLEGAL LIBRARY SPECIFIED</p> <p>In the :PREPRUN or :RUN command, the LIB=library parameter was not specified correctly—only <i>S</i> (system), <i>P</i> (public), and <i>G</i> (group) are permitted entries for this parameter. Re-enter command with proper <i>library</i> requested.</p>

Table 10-1. Command Interpreter Error Messages (Continued)

ERRNUM	MESSAGE
64	<p>UNABLE TO ACCESS TERMINAL FILE</p> <p>Your session/job could not open \$STDIN. See system supervisor.</p>
66	<p>ILLEGAL DEVICE OR INVALID INPUT SPEED</p> <p>For the :SPEED command, an illegal <i>inspeed</i> parameter was specified for this device. Check this parameter against characteristics of device used.</p>
67	<p>ILLEGAL DEVICE OR INVALID OUTPUT SPEED</p> <p>For the :SPEED command, an illegal <i>outspeed</i> parameter was specified for this device. Check this parameter against the characteristics of the device used.</p>
68	<p>SUBQUEUE IS LINEAR AND HAS NO QUANTUM</p> <p>This command attempted to set a time-quantum for a linear subqueue, but quanta can only be changed for circular subqueues.</p>
69	<p>UNKNOWN SUBQUEUE</p> <p>The subqueue referenced by this command could not be identified by the system. Check names of subqueues available.</p>
70	<p>RIN CURRENTLY IN USE</p> <p>The :FREERIN command did not succeed because the RIN requested was in use. Wait until the RIN no longer is in use, and re-enter command.</p>
71	<p>RIN NOT ALLOCATED TO THIS USER</p> <p>The :FREERIN command attempted to de-allocate a RIN that does not belong to you; only the owner of the RIN can free it.</p>
72	<p>RIN TABLE FULL</p> <p>The :GETRIN command was issued, but no RIN's are presently available. Re-enter command when a RIN is available or see system supervisor.</p>
73	<p>DIRECTORY ERROR</p> <p>A miscellaneous error was detected in accessing the file directory; re-enter the command.</p>
74	<p>INVALID LIST FILE</p> <p>An improper list file was specified in the command—re-specify, naming another file.</p>
75	<p>UNDEFINED JOB NAME</p> <p>The specified job does not exist, or is in an improper mode for the command (for example, when the :TELL command is applied to a job not in execution).</p>
76	<p>MESSAGE ROUTING ERROR</p> <p>An internal routing error occurred; <i>detail</i> indicates the type of error. Inform system supervisor.</p>

Table 10-1. Command Interpreter Error Messages (Continued)

ERRNUM	MESSAGE
77	<p>TOO MANY FILESETS</p> <p>Too many <i>filesets</i> were specified in a :RESTORE command. The :RESTORE command requires the following limits for specifying <i>filesets</i>: 10 by account name; and 15 by account and group name; and 20 by account, group, and file name. More than one :RESTORE can be applied against the same tape, using fewer filesets than the limit in each command.</p>
78	<p>IMPROPER TAPE FORMAT</p> <p>A tape being restored (:RESTORE command) has a valid STORE/RESTORE label, but the information on the tape does not conform to the format of a tape written by the :STORE or :SYSDUMP commands. Ensure that a proper tape is used.</p>
79	<p>NON-EXISTENT USER</p> <p>The command specified a user not defined in the system. Check to ensure that the spelling of the user name, as recorded in the system, is correct. If it is and the user still cannot be referenced, see the system manager.</p>
80	<p>SPOOFLE I/O ERROR</p> <p>A disc input/output error occurred or the command was unable to successfully obtain spool file disc space. The :STREAM command discontinues processing.</p>
81	<p>STACK TOO SMALL</p> <p>The :STREAM command cannot obtain sufficient stack space to begin processing.</p>
82	<p>FACILITY NOT ENABLED</p> <p>The console operator has not enabled the :STREAM command facility.</p>
83	<p>BINARY FILE ILLEGAL</p> <p>The :STREAM command cannot spool binary files.</p>
100	<p>FILE SYSTEM ERROR</p> <p>An error was returned from the File System; <i>detail</i> shows the actual File System Error Number, interpreted by referring to File System Error Codes 10 through 110 described in <i>MPE Intrinsic Reference Manual</i>. If <i>detail</i> is not displayed, an unexpected end-of-file condition was detected.</p>
101	<p>UNIT NOT READY</p> <p>The command specified an input/output unit that was not ready. Request the operator to ready the unit.</p>

Table 10-1. Command Interpreter Error Messages (Continued)

ERRNUM	MESSAGE
102	<p>NO WRITE RING</p> <p>The command specified, as an output device, a tape unit with no write-enable ring on the tape. Request the operator to insert the ring.</p>
103	<p>INCONSISTENT FILE OPERATION</p> <p>The command was inconsistent with the access type, record type, or device type for the referenced file. Check for proper consistency.</p>
104	<p>PRIVILEGED FILE VIOLATION</p> <p>The command attempted unauthorized access to a privileged file.</p>
105	<p>INSUFFICIENT DISC SPACE</p> <p>The command requested more disc space than that available. If possible, resubmit command with less space requested or purge existent files to make available additional space.</p>
106	<p>NON-EXISTENT ACCOUNT</p> <p>The command referenced an account not defined in the system. See system manager.</p>
107	<p>NON-EXISTENT GROUP</p> <p>The command referenced a group not defined in the system. See account manager.</p>
108	<p>NON-EXISTENT FILE</p> <p>The command referenced a file not defined in the session/job temporary or system file domain. Ensure that the file name was entered correctly.</p>
109	<p>INVALID FILE NAME</p> <p>The command requested a file with an invalid name. Check the file name spelling and syntax.</p>
110	<p>DEVICE UNAVAILABLE</p> <p>The command referenced a device that is not available. Request availability from the operator, or re-specify the device.</p>
111	<p>INVALID DEVICE SPECIFICATION</p> <p>The command requested a device with an invalid device specification. Check the specification used, and re-enter command.</p>
112	<p>NO PASSED FILE</p> <p>The command required a passed file, but no file was passed.</p>

Table 10-1. Command Interpreter Error Messages (Continued)

ERRNUM	MESSAGE
113	<p>EXCLUSIVE VIOLATION</p> <p>The command required exclusive access to a file already being accessed, or required access to a file to which another program had exclusive access.</p>
114	<p>LOCKWORD VIOLATION</p> <p>The command requested a file protected by a lockword, but no lockword (or an incorrect lockword) was supplied.</p>
115	<p>SECURITY VIOLATION</p> <p>The command requested a file protected against this access by the MPE File Security System.</p>
116	<p>DUPLICATE NAME</p> <p>The attempted creation of an account, group, user, permanent file or temporary file encountered a duplicate name. For files, this could be a duplicate name in the system or session/job temporary file directory.</p>
117	<p>DIRECTORY OVERFLOW</p> <p>The command caused the session/job temporary file directory or system directory to overflow.</p>
118	<p>ATTEMPT TO SAVE SYSTEM FILE AS JOB TEMPORARY</p> <p>The command attempted to save a system file in the session/job temporary file directory. Re-enter the command correctly.</p>
119	<p>IN USE: CAN'T BE PURGED</p> <p>The command attempted to purge a file, group, user, or account that was in use. Wait until the item referenced is free, and re-enter the command.</p>
120	<p>CREATOR CONFLICT</p> <p>The :RENAME, :SECURE, :RELEASE, or :ALTSEC command was entered for a file by someone other than the creator of the file; use of these commands is restricted to the file creator.</p>
121	<p>GROUP FILESPACE EXCEEDED</p> <p>The maximum disc space allocatable for the group's files was exceeded.</p>
122	<p>ACCOUNT FILESPACE EXCEEDED</p> <p>The maximum disc space allocatable for the account's files was exceeded.</p>
200	<p>CREATE ERROR</p> <p>An error has been detected by the CREATE intrinsic; <i>detail</i> indicates the actual CREATE intrinsic error number, described in <i>MPE Intrinsic Reference Manual</i>, under <i>CREATE Errors</i>.</p>

Table 10-1. Command Interpreter Error Messages (Continued)

ERRNUM	MESSAGE
201	<p>LOAD ERROR</p> <p>An error has been detected by the LOAD intrinsic; detail shows the actual Loader error number, described in <i>MPE Intrinsic Reference Manual</i>, under <i>Loader Errors</i>.</p>
202	<p>ILLEGAL LIBRARY SEARCH</p> <p>The command requested an illegal library search.</p>
203	<p>UNKNOWN ENTRY POINT</p> <p>The command specified a program entry point that could not be located.</p>
204	<p>DATA SEGMENT TOO LARGE</p> <p>The command required a data segment such that either:</p> <ol style="list-style-type: none"> 1. The data segment exceeded the system-defined maximum size allowed; or 2. The data segment required, or the <i>maxdata</i> parameter specified, exceeds 32767 words.
206	<p>DATA SEGMENT LARGER THAN MAXDATA SPECIFICATION</p> <p>The command required a data segment larger than the maximum size specified by yourself in this case.</p>
207	<p>ILLEGAL NUMBER OF CODE SEGMENTS</p> <p>The command required more than 152 code segments or more than the maximum allowed by the system.</p>
208	<p>INVALID PROGRAM FILE</p> <p>The command referenced a program file whose filecode or format is incorrect.</p>
209	<p>CODE SEGMENT TOO LARGE</p> <p>The command required a code segment larger than the maximum size permitted by the system.</p>
210	<p>MORE THAN ONE EXTENT IN PROGRAM</p> <p>The command referenced a program containing more than one disc extent.</p>
214	<p>SYSTEM SL ACCESS ERROR</p> <p>A system segmented library access error occurred; <i>detail</i> specifies a File System Error Code (Error Codes 10 through 110) described in <i>MPE Intrinsic Reference Manual</i>. If <i>detail</i> is not displayed, an unexpected end-of-file condition was detected.</p>
215	<p>PUBLIC SL ACCESS ERROR</p> <p>A public segmented library access error occurred; <i>detail</i> specifies a File System Error Code (Error Codes 10 through 110) described in <i>MPE Intrinsic Reference Manual</i>. If <i>detail</i> is not displayed, an unexpected end-of-file condition was detected.</p>

Table 10-1. Command Interpreter Error Messages (Continued)

ERRNUM	MESSAGE
216	<p>GROUP SL ACCESS ERROR</p> <p>A group segmented library access error occurred; <i>detail</i> specifies a File System Error Code (Codes 10 through 110) described in <i>MPE Intrinsic Reference Manual</i>. If <i>detail</i> is not displayed, an unexpected end-of-file condition was detected.</p>
217	<p>PROGRAM FILE ACCESS ERROR</p> <p>A program file access error occurred; <i>detail</i> specifies a File System Error Code (Codes 10 through 110) described in <i>MPE Intrinsic Reference Manual</i>. If <i>detail</i> is not displayed, an unexpected end-of-file condition was detected.</p>
218	<p>LIST FILE ACCESS ERROR</p> <p>A list file access error occurred; <i>detail</i> specifies a File System Error Code (Codes 10 through 110) described in <i>MPE Intrinsic Reference Manual</i>. If <i>detail</i> is not displayed, an unexpected end-of-file condition was detected.</p>
219	<p>INVALID SYSTEM SL FILE</p> <p>The command referenced the system segmented library file, whose filecode or format is incorrect.</p>
220	<p>INVALID PUBLIC SL FILE</p> <p>The command referenced a public segmented library file whose filecode or format was incorrect.</p>
221	<p>INVALID GROUP SL FILE</p> <p>The command referenced a group segmented library file whose filecode or format was incorrect.</p>
222	<p>INVALID LIST FILE</p> <p>The command referenced an invalid list file.</p>
223	<p>ILLEGAL PROGRAM DEALLOCATION</p> <p>The command illegally attempted to deallocate a program.</p>
224	<p>ILLEGAL PROGRAM ALLOCATION</p> <p>The command illegally attempted to allocate a program.</p>
225	<p>ILLEGAL PROCEDURE DEALLOCATION</p> <p>The command illegally attempted to deallocate a procedure.</p>
226	<p>ILLEGAL PROCEDURE ALLOCATION</p> <p>The command illegally attempted to allocate a procedure.</p>

Table 10-1. Command Interpreter Error Messages (Continued)

ERRNUM	MESSAGE
227	<p>ILLEGAL CAPABILITY</p> <p>The command referenced a file or program such that: permanent program file capability exceeds its group's; temporary file capability exceeds user's; or program requires privileged mode but program's capability does not include it.</p>
228	<p>UNABLE TO OBTAIN CST ENTRIES</p> <p>The command required code segment table (CST) entries that could not be obtained.</p>
229	<p>UNABLE TO OBTAIN PROCESS DST ENTRY</p> <p>The command required data segment table (DST) entries that could not be obtained.</p>
230	<p>UNABLE TO OBTAIN VIRTUAL MEMORY</p> <p>The command required virtual memory space that could not be obtained.</p>
231	<p>TRACE SUBSYSTEM NOT PRESENT</p> <p>The command requested loading of a program to be traced, but the TRACE subsystem is not in the system segmented library as required.</p>
232	<p>PROGRAM LOADED IN OPPOSITE MODE</p> <p>The command requested that a program be simultaneously loaded in NORMAL and NOPRIV mode.</p>
250	<p>SEGMENTER ERROR</p> <p>A Segmenter error was detected in attempting to prepare a program; <i>detail</i> may show additional information, as follows:</p> <ul style="list-style-type: none"> 3 = Unable to create Segmenter process. 4 = Unable to access Segmenter process. 5 = Unable to send mail to Segmenter process. 6 = Unable to receive mail from Segmenter process. 7 = Segmenter process aborted.
900	<p>EOF ON \$STDIN</p>
901	<p>I/O ERROR ON \$STDIN</p>
950	<p>EOF ON \$STDLIST</p>
951	<p>I/O ERROR ON \$STDLIST</p>

COMMAND INTERPRETER WARNING MESSAGES

When certain unusual but generally non-fatal conditions arise during interpretation or execution of an MPE command, various warning messages may appear on the session/job list device. Most such messages do not interrupt sessions nor terminate batch jobs; a session/job generally continues immediately following the warning message. The message itself appears simply as:

message

The Command Interpreter Warning Messages are summarized in table 10-2.

NOTE:

In addition to the messages listed in table 10-2, other Command Interpreter Warning Messages are also output. Table 10-2 shows only those messages that require explanation: the other messages are self-explanatory.

Table 10-2. Command Interpreter Warning Messages

CAPABILITY EXCEEDS ACCOUNT'S

Your capability is greater than your account's capability. You are given the maximum capability possible—the intersection of the two sets—for the session/job.

COMMAND IGNORED—NOT ALLOWED IN BREAK

This command cannot be executed during a break, without aborting the job.

DEFAULT VALUES TAKEN

Some default values were taken during loading of your program (during execution of commands such as :RUN, :SPLGO, and :PREPRUN, for example). These conditions are described under the discussion of the CREATE intrinsic (CCG condition code) in *MPE Intrinsic Reference Manual*.

EXECUTION PRIORITY REQUESTED EXCEEDS CAPABILITY

The scheduling priority requested at log-on time through the :PRI= parameter of the :HELLO or :JOB command, or its default value, is greater than your maximum priority as defined by the account manager. You are given the highest priority allowed below BS that satisfies your maximum priority restriction.

INVALID RESPONSE

You gave an invalid response to a query from MPE requiring only a YES or NO response; you are given a second chance to respond.

MAXPRI EXCEEDS ACCOUNT'S

Your maximum priority value (as defined by the account manager) is greater than that of the account. Your session/job is given, as its maximum priority, the smaller of the two values.

MESSAGE TOO LONG

The *message* parameter for the :TELL or :TELLOP command is too long to permit the destination message (including its prefix) to fit within a single 72-character line. (The total length includes the prefix, which in turn includes the fully-qualified session/job name of the sender.) The message is not sent.

PREMATURE JOB TERMINATION

An error in a batch job command (without a preceding :CONTINUE command) caused the job to fail without further command interpretation.

READ PENDING

A *break* request was initiated while a read was pending for your program. The read must be satisfied following entry of the :RESUME command. All characters entered for a partial record before the *break* are retained.

USER NOT ACCEPTING MESSAGES

The session/job to receive your message, though defined, is not in EXECUTION state.

SYSTEM MESSAGES

Miscellaneous conditions that terminate or otherwise affect sessions/jobs are reported through system messages, shown in table 10-3. These messages may appear at various times during the course of a running session/job on the standard list device.

Table 10-3. System Messages

CAN NOT INITIATE NEW SESSIONS NOW

New sessions cannot be initiated due to one of the following problems:

1. Insufficient system resources to start session.
2. Session limit would be exceeded.
3. Your input priority (INPRI =) is not greater than current jobfence.

NOTE: System managers and system supervisors can bypass rejections due to 2 and 3, above, by supplying HIPRI in :HELLO command.

* { SESSION
JOB } ABORTED BY SYSTEM MANAGEMENT *

The session/job has been aborted by the computer operator or system supervisor through the appropriate command. An immediate log-off then takes place.

* { SESSION
JOB } HAS EXCEEDED TIME LIMIT *

The session/job has exceeded the time limit specified in the TIME=parameter of the :HELLO or :JOB command. An immediate log-off then takes place.

WARNING: PRIORITY = xxx

The priority passed to the CREATE intrinsic resulted in a conflict with another process, and the priority then assigned was **xxx** instead of the requested value.

LMAP NOT AVAILABLE

An LMAP of the process being created, or program file being :RUN, is not available because the code segments are already loaded.

****POWER FAIL****

Power failure has occurred and automatic restart is in progress. It is possible that a character has been lost due to a transmission error when the power failure occurred.

TERMINALS SUPPORTED BY MPE

APPENDIX

A

TERMINAL TYPE	DESCRIPTION
0	ASR-33 EIA-compatible (HP 2749B) Terminal (10 cps).
1	ASR-37 Teleprinter Terminal with Paper Tape Reader/Punch (10 cps).
2	ASR-35 EIA-compatible Terminal (10 cps).
3	Execuport 300 Data Communications Transceiver Terminal (10/15/30 cps).
4	HP 2600A or DATAPOINT 3300 Keyboard-Display Terminal (10/15/30/60/120/240 cps).
5	Memorex 1240 Communication Terminal (10/15/30/60 cps). Note: This terminal must be equipped with even parity-checking option.
6	General Electric Terminet 300 or 1200, or Data Communications Terminal, Model B (10/15/30/120 cps) with Paper Tape Reader/Punch, Option 2. Note: This terminal must be equipped for "ECHO PLEX."
9	HP 2615A Terminal (10/15/30/60/120/240 cps).
10	HP 2640A/2644A Interactive Display Terminal (Primarily character mode.) (10-240 cps.)
11	HP 2640A/2644A Interactive Display Terminal (Primarily block mode.) (10-240 cps.)

DETAILS OF PROGRAM EXECUTION

APPENDIX

B



In MPE, programs are run on the basis of *processes* created and handled by the system. A process is not a program itself, but the unique execution of a program by a particular user at a particular time. Therefore, if the same program is run by several users, or more than once by the same user, it is executed by several distinct processes.

The process is the basic executable entity in MPE. A process consists of a process control block (PCB) that defines and monitors the state of the process, a dynamically-changing set of code segments, and a data area (stack) upon which these segments operate. Thus, while a *program* consists of instructions (not yet executable) and data in a file, a *process* is an executing program with a data stack assigned. The code segments used by a process can be shared with other processes, but its data stack is private (Figure B-1). For example, each user working on-line through the BASIC language is running his program under a separate process; all use the same code (the only copy of the BASIC interpreter in the system), but each has his own stack.

Processes, and the elements that comprise them, are invisible to the programmer accessing MPE through standard capabilities; in other words, this programmer has no control over processes or their structure. For this user, MPE automatically creates, handles, and deletes all processes. The user with certain optional capabilities, however, can create and manipulate processes directly. See *MPE Intrinsic Reference Manual* for further details.

OVERVIEW OF ALLOCATION/EXECUTION

When a program is run, it is allocated and executed. Before allocation/execution takes place, MPE checks to verify the following points:

- For *program files that are permanent files*, the capabilities assigned to the program must not exceed those assigned to the group to which the program file belongs; otherwise, an error occurs.
- For *program files that are temporary files in the session/job temporary file domain*, the capabilities assigned to the program must not exceed those of the user running the program; otherwise, an error occurs.
- For *privileged segments, when the NOPRIV parameter is omitted from the :RUN (program execution) command*, the capabilities assigned to the program must include the privileged mode capability for the segment to be loaded in privileged mode; otherwise, an error occurs.

During *allocation*, the MPE loader binds the segments from the program file to referenced external segments from a segmented library (SL). Then, the first code segment to be executed and the stack are moved into main memory. *Execution* now begins. As it progresses, many processes may be created, run, and deleted. For each process in execution, one or more code segments and one stack, operating under control of a process control block (PCB), typically exist in main memory. Not all code segments belonging to a process in execution need exist in main memory simultaneously. Typically, a process operates on a set of segments that are dynamically swapped between memory and disc. MPE main-

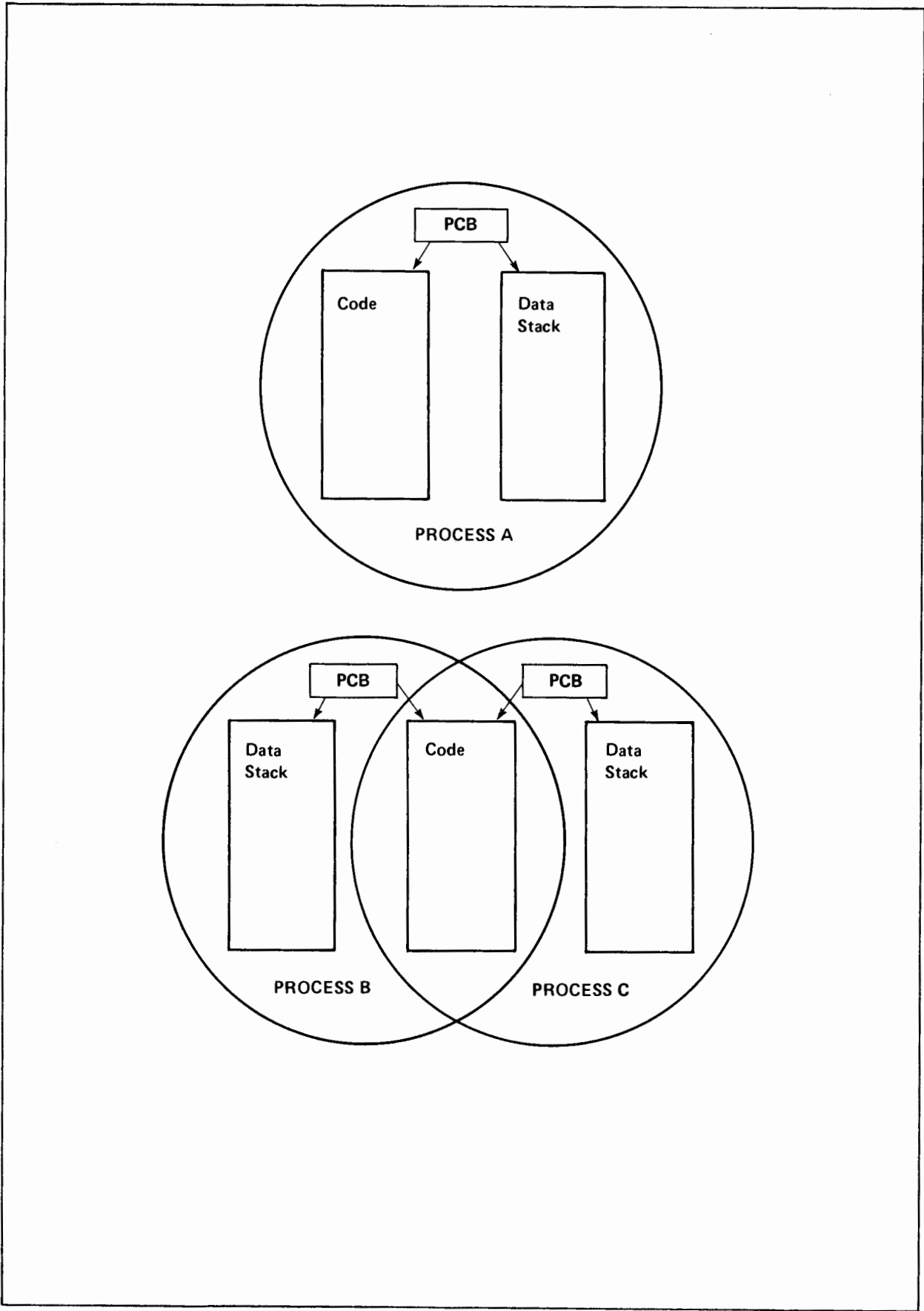


Figure B-1. Code-Sharing and Data Privacy

tains records of the frequency each segment is used, so that those used least frequently in main memory become the most eligible for swapping out. You never need to determine whether a segment is in main memory or on disc at any given time — this is always done for you automatically by MPE.

During allocation/execution, MPE keeps track of each code segment by maintaining information about its nature and current location in the code segment table (CST). Similarly, information about the stack is dynamically recorded in the data segment table (DST).

A particular program can be run by many user processes simultaneously, with all processes accessing the same copy of sharable code. But unlike the program code segments, the data segment containing the stack is private to each user's process and cannot be shared with others. As execution progresses, data enters and leaves the stack dynamically. Within the stack, data is arranged as a linear group of items accessed from one end (called the top-of-stack). When the last instruction in a process is executed, MPE releases those segments associated only with that process, including the stack segment, and deletes their related entries in the CST and DST. (However, shared code segments are not released until the last process using them is deleted.) All files opened by this process are closed.

PCB/CODE SEGMENT/STACK INTERACTION

Each process control block (PCB) contains information needed to control a process. This information includes the priority of the process and pointers to ancestor and descendent processes. (See *MPE Intrinsic Reference Manual* for a discussion of inter-process relationships.)

Each code segment executed by a process can contain one or more program units that include calls to procedures elsewhere in this segment or in another segment. When a code segment is executing in main memory, it is defined by pointers in three hardware registers: the Program Base (PB), Program Counter (P), and Program Limit (PL) registers (Figure B-2). There is only one set of these registers; at any particular instant, their contents refer to the code segment currently in execution. The PB register contains the absolute address of the starting location of the segment in main memory. The P register holds the absolute address of the instruction currently being executed. The PL register indicates the absolute address of the last location of the code segment. Execution can only be transferred from this segment by an interrupt or by a call or exit instruction referencing a procedure in another segment, in which case the PB, P, and PL registers are reset to reflect the characteristics of the new segment. Whenever an instruction is executed, the PL and PB registers are checked to ensure that referenced addresses fall within the proper segment boundaries. This *bounds check* guarantees that other programs and the system itself are protected against improper access. Since all addresses within a code segment are relative to the contents of the three program registers, a segment can be relocated anywhere in main memory and only the register contents and CST need be changed to reflect this transfer.

As with code segments, there are normally several stacks present in memory. (One stack exists for each process.) But since the execution of any process is interleaved with that of others, only one stack is active at any particular instant. Most dynamic computational operations take place on the stack. The last element added to the stack is placed in the word at the *top-of-the-stack*. In this position, it is the first element removed when the process associated with the stack requests data from the stack. (In other words, the last element in is the first one out.) Each time data is added to the stack, the previous top element becomes the second element from the top; each time the topmost element is removed, the second element returns to the top of the stack.

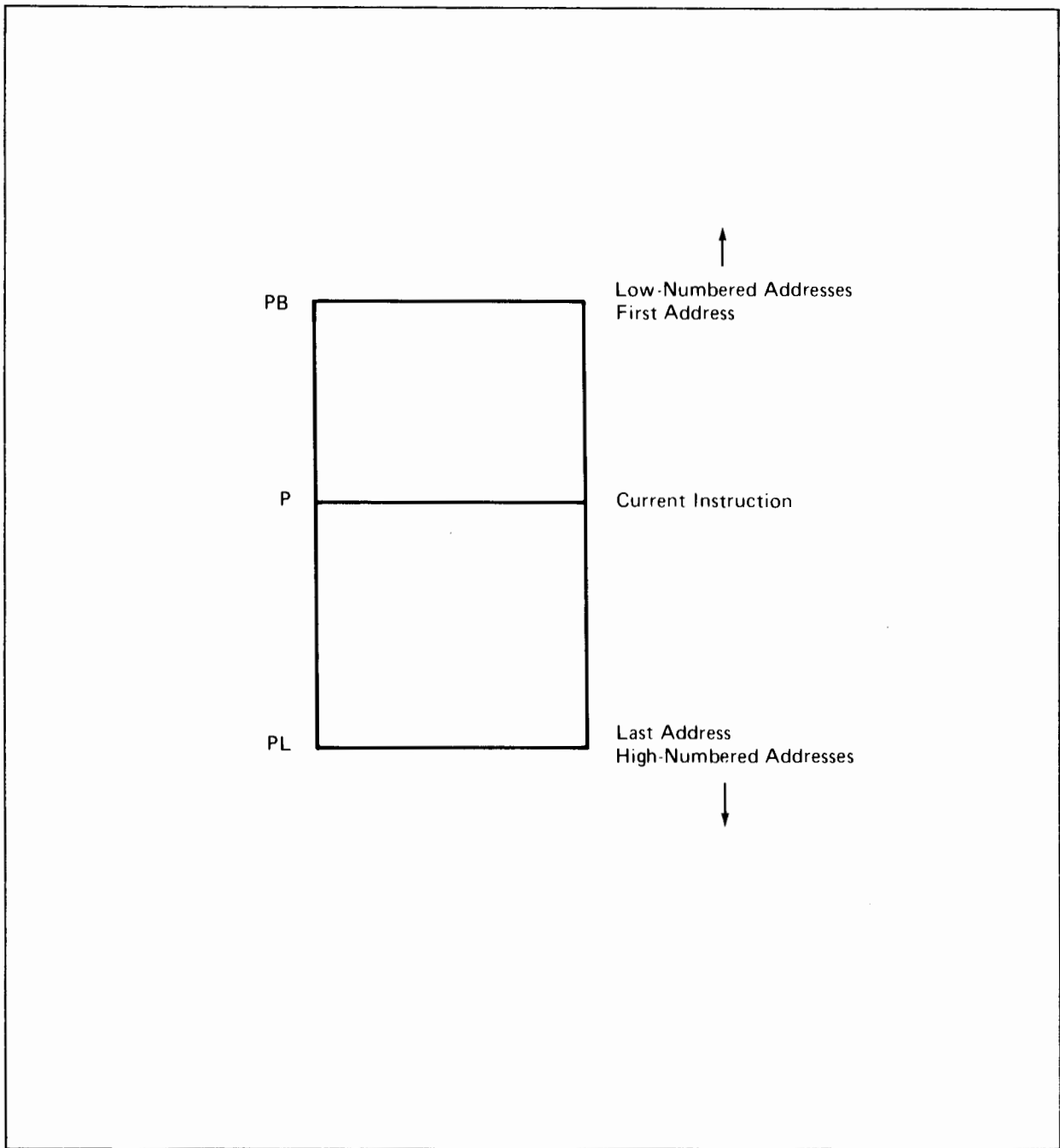


Figure B-2. Code Segment and Associated Registers

Programmers operate *directly* on elements in the stack only when using the SPL language. However, any program in any other language references the stack implicitly when it manipulates data, although the user need not be aware of this. The principal data areas within the stack segment and their purposes are summarized in table B-1. (The values of specific constants appearing in table B-1 are currently accurate but may be changed in future releases of the operating system.) The boundaries of these areas are delimited by the addresses stored in the registers indicated along the left side of the stack diagram appearing in figure B-3 (DST, DL, . . . Z registers). The PCBX, DL, and working stack areas are all subject to dynamic expansion or contraction as the process runs.

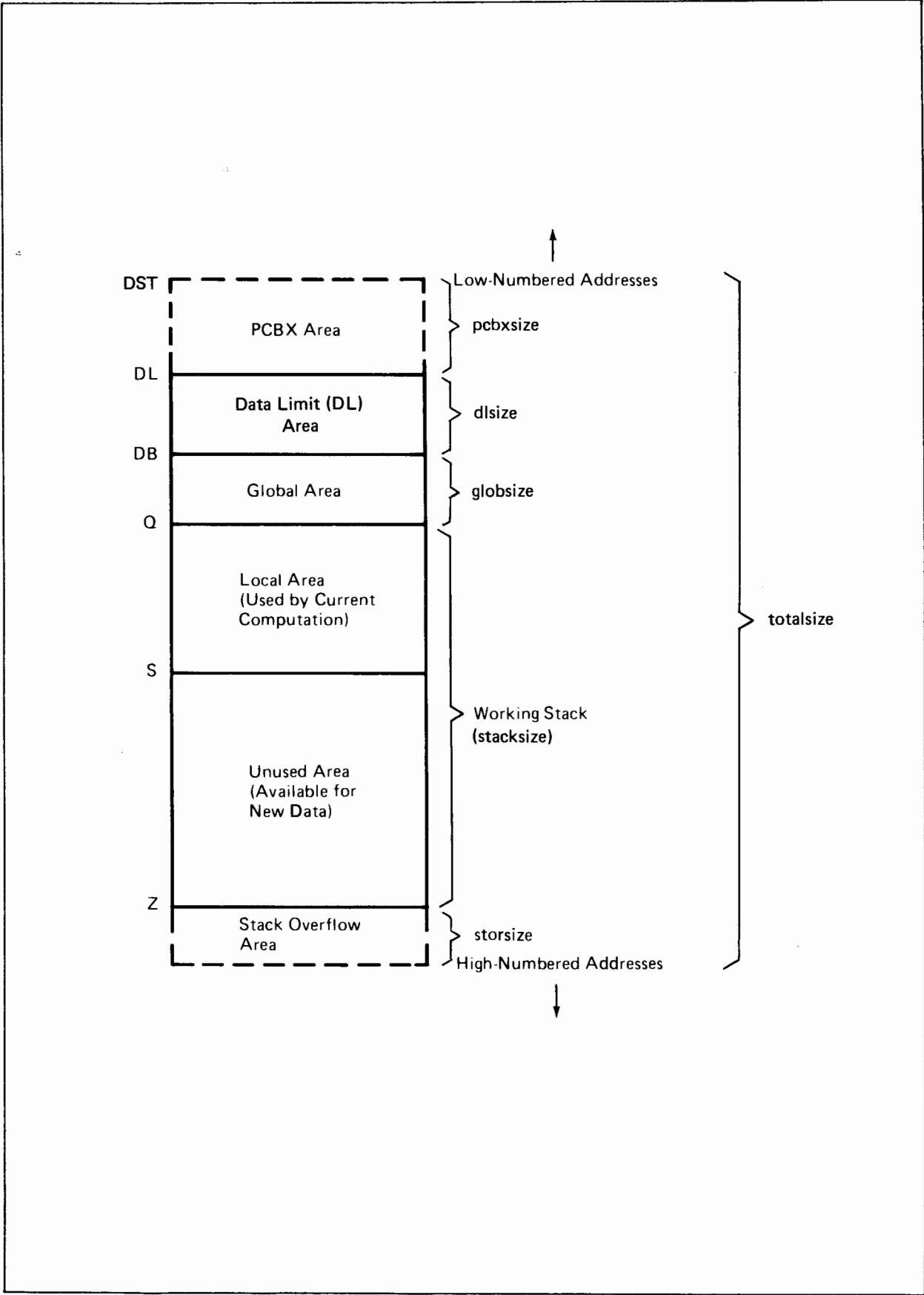


Figure B-3. Data (Stack) Segment and Associated Registers

Table B-1. Data Areas in Stack Segment

AREA	FUNCTION
PCBX	<p>The Process Control Block Extension (PCBX) area contains information necessary for the system to control process activity efficiently, such as register settings and file pointers. In general, the PCBX is used when the process is running in main memory; the PCB is used when it is not. Initially, this area is %402 words long.</p>
DL	<p>This area contains user-managed information accessed and used only through SPL programs such as compilers. These programs sometimes require this space for buffers. This area is bounded by the addresses stored in the data limit (DL) and data base (DB) registers, and is accessed via DB-negative addressing. (A portion of the DL area, that between the addresses DB-10 and DB-1, is reserved for data used by subsystems during their operation.) The initial size of the DL area (<i>dsize</i>) may be specified in the :RUN command or CREATE intrinsic. If omitted from the :RUN command, <i>dsize</i> is taken from the specifications supplied with the :PREP command. If not supplied anywhere by user, <i>dsize</i> is assigned a value of zero. The <i>dsize</i> actually granted is calculated as follows:</p> $dsize \leftarrow \{ [(specified\ dsize) + \%613] \text{ and } - 128 \} \%402$ <p>Note: In this calculation, <i>and</i> indicates a <i>logical-and</i> operation.</p> <p>This calculation assures that the area from the beginning of the segment to the DB address is an integral number of disc sectors. The resulting <i>dsize</i> is checked to ensure that it is less than 32768 words.</p>
Global	<p>This area is used for global variables; these are variables declared within the data group of a main program and usable by any procedure within that program. It also contains global arrays and pointers to those arrays. The size of this area (<i>globsize</i>) is set during program preparation and entered in the program file. This size, and the content of the global area, are determined by the number of global variables and their initialization as specified by the programmer. The area is delimited by the contents of the DB register and the stack marker (Q) register.</p>
Working Stack	<p>From the standpoint of user programs, this is the most active area of the stack—it is the area where the user's temporary data is stored and manipulated. It is bounded by the addresses indicated by the initial contents of the Q register (Q-initial) and the Z-register. Its size (<i>stacksize</i>) is specified in the :RUN command or CREATE intrinsic, or else is taken from the :PREP command specifications. By default, in the absence of such information, <i>stacksize</i> is supplied by MPE. (This default <i>stacksize</i> is specified by the System Supervisor when he configures the system.) The <i>stacksize</i> must be greater than 512 words but less than 32768 words.</p>
Local	<p>Within the working stack area, the local area contains data relating only to the procedure currently in execution. The current contents of the Q-register denotes the beginning of this area and the contents of the top-of-stack (S) register indicates the end, pointing to the last item of data in the stack.</p>
Unused	<p>Also within the working stack area, the unused area is the space that remains available for new data. This area is bounded by the addresses in the S register and stack limit (Z) register. The Z register indicates the last main-memory location that can be used by user data in the stack area.</p>

Table B-1. Data Areas in Stack Segment (Continued)

AREA	FUNCTION
Stack Overflow	This area is available for stack-overflow, a condition that occurs when the S-register pointer must be moved beyond the Z-register pointer and space must be provided for certain end-of-stack information. This buffer area lies between the Z-register contents and the end of the data segment. This area is currently %200 words long.

When the stack is allocated for a process, its total size (*totalsize*) is calculated as follows:

$$totalsize \leftarrow pcbxsize + dlsiz e + globsize + stacksize + storsize$$

MPE ensures that *totalsize* is less than 32768 words and also less than or equal to *maxstack* (the maximum allowed stack size specified during system configuration). The entire size of the data segment allocated for the stack (*dssize*) is calculated as follows:

$$dssize \leftarrow (totalsize + 7) land - 4$$

The *dssize* is effectively rounded upward to an integral multiple of four, and increased to include the four-word trailing main-memory link of the segment when present in memory. The segmenter descriptor (*dstsize*) in the Data Segment Table (DST) is:

$$dstsize \leftarrow dssize/4$$

Another stack-related value, *maxdata*, is maintained by MPE in the PCBX and is used to limit automatic stack expansion by fixing the maximum stack size. It may be specified in the :PREP command and recorded in the program file. It may also be specified in the :RUN command or CREATE intrinsic, in which case it overrides the :PREP command specification (if any). If *maxdata* is omitted from both :PREP and :RUN (or CREATE), MPE equates *maxdata* to *totalsize*. When specified, *maxdata* must be a positive number or zero. If the specified value exceeds the *maxstack* value defined during configuration, *maxdata* is equated to *maxstack*. During process execution, if the distance from the start of the data segment to the Z-address exceeds *maxdata* because of stack overflow, the process is aborted.

The size of the data segment as it exists in virtual memory on disc (*vsiz e*) is calculated as follows:

$$vsiz e \leftarrow dssiz e + 1536 \text{ words}$$

The maximum virtual-memory space allowed (*maxsiz e*) is:

$$maxsiz e \leftarrow \max(dssiz e, maxdata) + 1536$$

The above calculation includes three disc sectors for possible expansion within the PCBX area, and nine sectors for space used during possible stack-overflow abort. These extra twelve sectors, totaling 1536 words, imply a limit on *maxdata* of 31224 words.

Through bounds checks, MPE and certain hardware provisions ensure that data for the process remains within the limits defined by the contents of the DL and Z registers, and that no other user accesses this area. All locations are addressed relative to the DB, Q, and S registers.

Although the top-of-stack is logically indicated by the S register, up to four of the topmost elements of the stack can actually exist in central processor registers (the TR registers) rather than in main memory — in effect, the stack “spills over” from memory into these registers. This function is managed by the hardware and greatly enhances processing speed.

Before a process begins execution, stack space is first reserved for global data, beginning at the DB-address and terminating at the Q-address, which denotes the beginning of the dynamic *working stack* (figure B-4A). At this time, no data is stored beyond the Q-address, and so the S-register also points to the same address as the Q register — that is, the bottom and top of the working stack coincide. But, as the process begins execution, data is added to the stack, and the S-pointer (top-of-stack) moves away from the Q-pointer (figure B-4B). If, at some point, the process encounters a procedure call, a new area for data local to that procedure must be defined. To do this, the system hardware places a group of four words called the *stack marker* on top of the stack to save information necessary to re-create the currently-defined local area later. The Q and S registers are then pointed to the top word of the stack marker, which also delimits the beginning of a new, fresh and unique local area for the procedure just called (figure B-4C).

The words in the stack marker preserve the state of the machine at the time of the procedure call. These words contain the following information, (shown in order ascending toward the Q-address):

Word	Contents
Q-3	Current contents of the Index (X) register.
Q-2	The return address for the code segment, denoted by P+ 1 (relative to the PB register).
Q-1	The current contents of the Status register (which includes the number of the code segment containing the calling procedure).
Q-0	The delta-Q value, which is the number of words between the new and previous Q-locations, enabling the later re-setting of Q to its old value.

As data is added to the stack during execution of the new procedure, the S-pointer moves away from the new Q-pointer, reflecting the latest data added (figure B-4D). When the procedure exits back to the main program, the new local data area is deleted from the stack, the stack marker is used to restore the Q-pointer to its previous setting, any value returned by the procedure is left at the new top-of-stack, and the S-pointer is set to indicate the new top-of-stack (figure B-4E). This results in a “clean” stack from which temporary data local to the called procedure is eliminated because it is no longer needed.

Whenever a procedure is called, the Q and S registers are manipulated in this manner. The Q register changes with each procedure call and exit; the S register may change when an instruction references data. Thus, when a process executes a main program (outer block) that calls three procedures, there will be a maximum of four local areas (one for the main program and three for the procedures called by it) on the stack. Each procedure’s local area is delimited at its base by its own stack marker. Within these stack markers, the delta-Q words form a logical chain that links the present Q register setting back to its initial value.

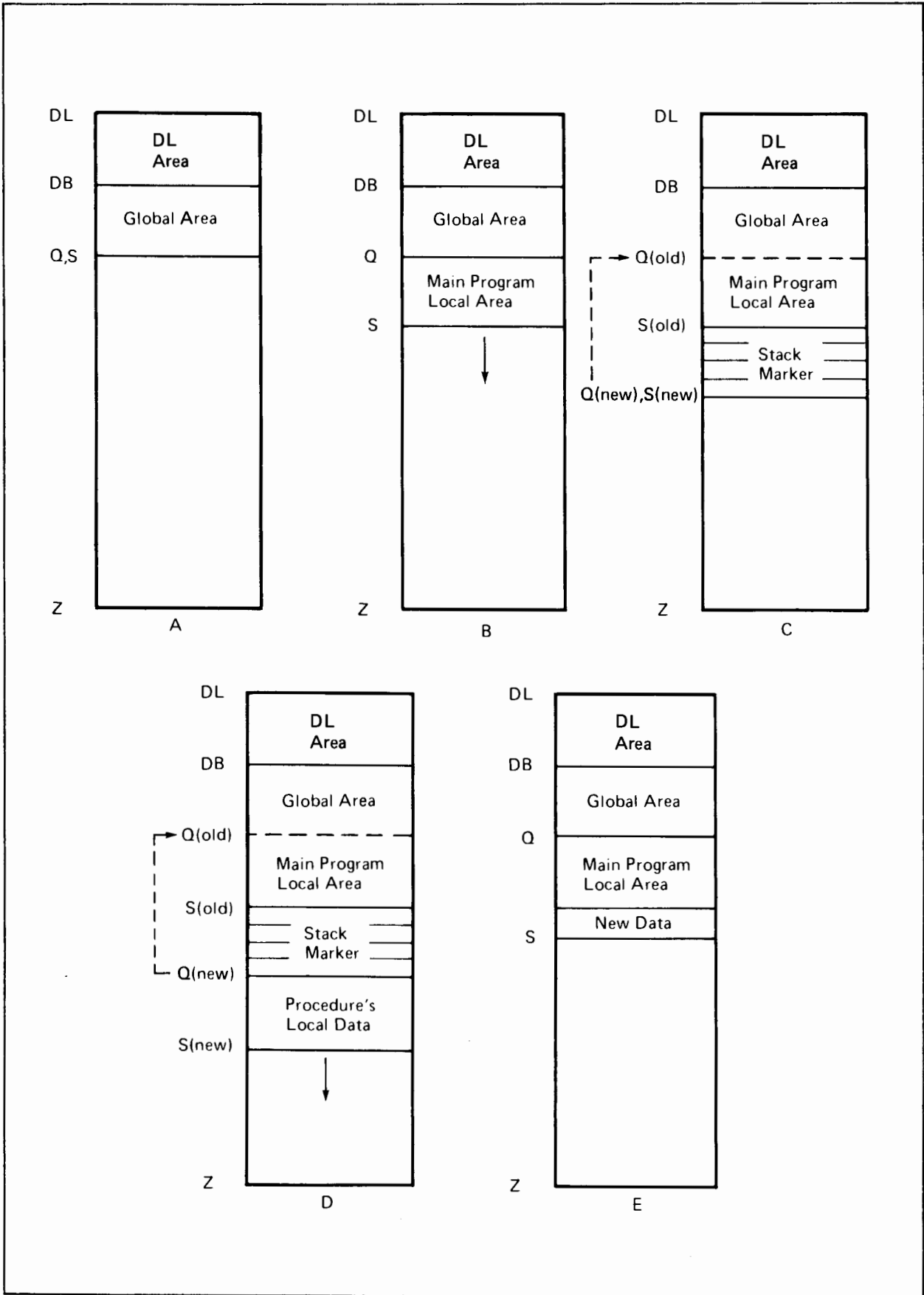


Figure B-4. Stack Operation

A

Abnormal session termination, 3-22
 :ABORT command, 2-5, 3-16, 3-18, 3-19
 Aborting commands, 2-5, 3-16, 3-17
 Aborting programs, 2-5, 3-18, 3-19
 =ABORTJOB console command, 3-22, 4-19
 Access modes for files, 6-36-6-38, 6-78-6-81
 Access restrictions on files, 6-44, 6-78-6-81
 Accessing files, 2-50
 Account names, 2-65, 2-69, 3-5, 4-4, 6-19
 Accounting information, 2-95, 3-14, 5-6-5-8, 6-42
 Accounts
 security restrictions on, 6-76-6-77
 System (SYS), 6-76
 ACTIVE devicefile state, 3-21, 3-22, 4-17-4-19
 Actual file designator, 6-24-6-18
 AFT, 6-5, 6-7
 Allocating programs, 7-1, 7-3, B-1, B-3
 =ALTFILE console command, 4-17
 =ALTJOB console command, 4-7, 5-1
 :ALTSEC command, 1-1, 2-7, 2-8, 6-80
 Anticipatory reading, 6-34
 ASCII - coded files, 6-32
 Available File Table, 6-5, 6-7

B

Back-referencing files, 6-22, 7-16
 :BASIC command, 2-9, 7-14
 BASIC compiler, 2-11-2-16, 7-15
 BASIC interpreter, 2-9, 3-20, 7-14, 7-15
 BASIC SAVE-FAST files, 2-11-2-16, 7-15
 :BASICGO command, 2-13, 7-15
 :BASICOMP command, 2-11, 2-12, 7-15
 :BASICPREP command, 2-15, 2-16, 7-15
 Binary-coded files, 6-32
 Block mode for HP 2640/44 terminals, 3-29-3-31
 Blocking of records, 6-8, 6-41-6-44
 Blocks
 definition of, 6-1
 file control, 2-85
 relation to files and records, 6-10
 Bounds checking, 13-3
 Breakable commands, 3-15
 =BREAKJOB console command, 5-1
 Breakpoints, 2-35, 2-83
 Buffers, 2-49, 2-50, 6-8, 6-34, 6-35
 :BUILD command, 1-1, 2-11, 2-12, 2-15, 2-17-2-20, 2-24,
 3-15, 6-55, 6-60, 7-5, 7-7, 9-6
 :BYE command, 2-21, 3-4, 3-14, 3-22

C

Capability-class attributes, 2-80, 2-84
 Card-reader/punch unit, 2-41
 Carriage-control characters, 6-38, 6-39
 CAUSEBREAK intrinsic, 2-1
 Central processor, time limits and usage accounting for,
 2-67, 2-71, 2-95, 3-1, 3-4, 3-10, 3-13, 3-14, 3-22, 4-1,
 4-6, 4-7, 4-10, 5-6-5-10
 Character mode for HP 2640/44 terminals, 3-29
 Class name, device, 2-18, 2-51, 2-71, 2-102, 6-25, 6-26, 8-1
 :COBOL command, 1-1, 2-23, 2-24, 3-16, 7-4, 7-9
 COBOL compiler, 2-23-2-28, 3-16
 :COBOLGO command, 2-25, 7-8
 :COBOLPREP command, 2-27, 7-6
 Code segment table, B-3
 Code segments, B-1, B-2, B-3
 Codes, file, 2-19, 2-52, 6-39, 6-40, 7-9
 Command Interpreter, 1-2, 1-5, 2-35, 3-1-3-4, 3-23, 4-2
 Command intrinsic, 1-5, 2-1
 Commands
 breakable, 3-15
 delimiters for, 1-3
 elements in, 1-2, 2-4
 entry of, 1-2
 errors in, 1-5
 functional list of, 2-2, 2-3
 names in, 1-2, 1-3
 octal numbers in, 1-1
 parameter lists in, 1-2, 1-3
 programmatic execution of, 1-5
 prompt character for, 1-2, 1-3
 purposes of, 1-1
 :COMMENT command, 2-29, 4-16
 Comments in jobs, 4-16
 Comments in programs, 2-29
 Communicating with Console Operator, 9-1, 9-2, 9-4
 Communicating with other users, 9-1-9-4
 Compiler
 BASIC, 2-11-2-16, 7-15
 COBOL, 2-23-2-28, 3-16, 7-4, 7-6, 7-8, 7-9
 FORTRAN, 2-55-2-60, 3-2, 3-3, 4-1-4-3, 7-5
 RPG, 2-109, 2-114
 SPL, 2-145-2-150, 3-16
 Compiling programs, 7-1-7-9
 Connect-time, limits and usage accounting for, 2-95, 3-1,
 3-4, 3-13, 3-14, 4-2, 4-10, 5-6-5-8
 Console operator, 9-1, 9-2, 9-4
 Continuation characters, 1-5, 4-16
 :CONTINUE command, 2-31, 4-20, 4-21, 10-2
 \$CONTROL USLINIT compiler
 subsystem command, 3-3, 4-2, 4-3
 Copying files, 6-54
 Correcting characters on a terminal, 3-28
 Creating files, 2-17-2-20, 6-4-6-7, 6-55
 CST, B-3

D

Data
 end of, 2-39
 in devicefiles, 2-33
 read via :DATA command, 2-33, 3-19, 3-22, 4-19, 6-81-6-83
 spooling of, 2-153, 2-154, 6-81-6-83
:DATA command, 2-33, 3-19, 3-22, 4-19, 6-81-6-83
Data segment table, B-3
Data segments, B-1, B-2
Date, display of, 2-141
DB-Q (initial) area, 2-79, 2-83, B-5, B-6
:DEBUG command, 2-35
Debug facility, 2-35, 2-83, 2-115
=DELETE console command, 3-22, 4-21
Designators
 actual file, 6-14-6-18
 formal file, 2-47, 2-97, 6-4, 6-13-6-18, 7-17-7-19
Device class name, 2-18, 2-51, 2-71, 2-102, 6-25, 6-26, 8-1
Device Directory, 6-6, 6-7
Devices
 availability of, 8-1
 for files, 6-25, 6-26, 8-1-8-3
 ownership of, 8-1
 reporting status of, 8-1-8-3
 unit record, 6-32
Devicefiles
 blocking of, 6-8
 data in, 2-33
 definition of, 6-3
 handling of, 8-1, 8-4-8-7
 input, 8-4-8-10
 old and new, 6-6, 6-7
 output, 8-10-8-15
 spooled, 8-4, 8-6, 8-7, 8-12
Direct access of files, 6-8
Disc extents, 2-19, 2-20, 2-53, 6-5, 6-10, 6-42-6-44
Disc files, 6-3, 6-8, 6-9, 6-40-6-41
Disc space, 6-33, 6-41-6-44
Disposition of files, 2-51, 6-11, 6-49
DL area, B-5, B-6
DL-DB area, 2-79, 2-80, 2-83, 2-84, 2-116, B-5, B-6
DST, B-3

E

Echo facility, 3-8, 3-9, 3-24, 3-25, 3-26
:EDITOR command, 2-37, 3-3, 3-16, 7-15
Editor subsystem, 2-37, 3-3, 3-16, 7-15
End-of-data indication, 2-39, 3-19, 3-20, 4-2, 4-4, 4-11, 4-12, 6-81-6-83, 7-5
End-of-file indicators, 2-41, 6-83, 6-84
Entry-points, program, 2-28, 2-115, 2-117, 7-12
:EOD command, 2-39, 3-19, 3-20, 4-2, 4-4, 4-11, 4-12, 6-81-6-83, 7-5
:EOF: command, 2-41, 3-22, 4-19
:EOJ command, 2-43, 4-2, 4-4, 4-10, 4-11, 4-19
Error messages, 3-23, 10-1-10-14

Errors

 during log-on, 3-13, 3-14
 in commands, 1-5
 in jobs, 4-10, 4-21, 10-2
 in sessions, 3-23, 10-2
Executing programs, 7-1, 7-3, 7-4, 7-8-7-13, B-1-B-9
Extents, 2-19, 2-20, 2-53, 6-5, 6-10, 6-42-6-44

F

Failure, power, 3-22, 4-8, 4-20
FCHECK intrinsic, 6-8
FCLOSE intrinsic, 6-3, 6-11, 6-79
FCOPY, 6-54, 6-61, 7-16
FGETINFO intrinsic, 6-8
File codes, 2-19, 2-52, 6-39, 6-40, 7-9
:FILE commands
 cancellation of, 6-54
 implied, 6-53, 7-16-7-19
 relationship to FOPEN parameters, 6-49, 6-50
 use of, 1-1, 2-45-2-54, 6-13-6-54, 7-7, 7-8, 7-10
File control blocks, 2-85
File designator, actual, 6-14-6-18
File Management System, 3-3
File Management System command summary, 6-85
File number, 6-7, 6-10
Filereference format, 6-18-6-21, 6-23
File space limit, 3-13, 3-14, 5-6-5-8
Files
 access modes, 6-36-6-38, 6-78-6-81
 access of, 2-50, 6-36-6-38, 6-78-6-81
 access restrictions on, 6-44, 6-78-6-81
 actual designators for, 6-14-6-18
 ASCII, 6-32
 back-referencing of, 6-22, 7-16
 binary, 6-32
 blocking of, 6-8, 6-41-6-44
 buffers for, 2-49, 2-50, 6-8, 6-34, 6-35
 card punch, header records for, 6-51, 8-11
 card punch, trailer records for, 6-51, 8-11
 carriage-control characters for, 6-38, 6-39
 characteristics of, 2-45-2-54, 6-5, 6-12-6-18
 conserving disc space for, 6-33
 copying of, 6-54
 creation of, 2-17-2-20, 6-4-6-7, 6-55
 default characteristics for, 6-50, 6-51
 definition of, 6-1
 deletion of, 2-89, 6-11, 6-62, 6-63
 device-dependent characteristics, 6-51, 6-52
 devices for, 6-25, 6-26, 8-1-8-3
 direct access of, 6-8
 disc consideration, 6-8, 6-9, 6-40-6-49
 disc space required for, 6-41-6-44
 disposition of, 2-51, 6-11, 6-49
 end of, 2-41, 3-22, 4-9, 6-83, 6-84
 exclusive access of, 6-45-6-47
 extents for, 2-19, 2-20, 2-53, 6-5, 6-10, 6-42-6-44
 formal designators for, 2-47, 2-97, 6-4, 6-13-6-18, 7-17-7-19



header records for, 6-51
input/output efficiency for, 6-33
labels for, 6-5, 6-17, 6-57, 6-60
line printer, header pages for, 6-51
line printer, special forms for, 6-53
line printer, trailer pages for, 6-51
list, 2-11, 2-23, 2-25, 2-27, 2-37, 2-55, 2-57, 2-59, 2-107,
2-109, 2-111, 2-113, 2-145, 2-147, 2-149
listing descriptions of, 2-75-2-77, 6-40, 6-55-6-57
lockwords for, 2-17, 2-47, 2-93, 6-12, 6-18, 6-20, 6-21,
6-60
multi-access of, 6-45-6-48
names of, 6-18-6-23
new, 2-45, 2-47, 6-5, 6-6, 6-11, 6-24, 6-55
number of copies output, 6-26
old, 2-24, 2-47, 6-7, 6-11, 6-24
opening, 6-4-6-7
outfence for, 6-26
output priority, 2-52
padding with blanks and zeros, 6-34
passing, 6-21, 7-4
permanent, 2-20, 2-75-2-77, 2-89, 2-91, 2-93, 2-119,
2-120, 3-15, 6-11, 6-12, 6-55, 6-66, 7-5, 7-7, 9-6
processing of, 6-3
program 2-27, 2-28, 2-57, 2-58, 2-79, 2-113, 2-114,
2-115-2-117, 7-3, 7-7, 7-9
purging of, 2-89, 6-11, 6-62, 6-63
renaming of, 2-93, 6-60
restoring to system, 2-101-2-103, 6-71-6-74
restricting access to, 6-36-6-38
saving, 1-1, 2-119, 2-120, 3-4, 4-2, 4-3, 6-11, 6-63, 6-64,
7-4, 7-6, 7-8, 7-9
security provisions for, 2-7, 2-91, 2-121, 3-8, 6-12, 6-37,
6-61, 6-62, 6-74-6-81
sequential access of, 6-8, 6-9
session/job temporary, 6-11, 6-12
sharable access of, 6-45-6-47
size, determination of, 6-41-6-44
source text, 2-23, 2-25, 2-27, 2-55, 2-57, 2-59, 2-109,
2-111, 2-113, 2-145, 2-147, 2-149
space limit for, 6-42
storing off-line, 2-151, 2-152, 6-64-6-70
structure of, 6-10
system-defined, 2-46, 6-18, 6-22
temporary, 2-20, 2-89, 2-93, 2-119, 2-120
trailer records for, 6-51
transferring data to and from, 6-8-6-10
transferring of, 6-61-6-62
user-defined, 6-18, 6-19
Fixed-length records, 6-27, 6-28
FLOCK intrinsic, 6-79
FOPEN intrinsic, 6-3, 6-10, 6-13, 6-17, 6-18, 6-20, 6-49,
6-50, 6-79
Formal file designator, 2-47, 2-97, 6-4, 6-13-6-18, 7-17-
7-19
:FORTGO command, 2-55, 3-2, 3-3, 4-2, 4-3, 7-8, 9-9
:FORTPREP command, 2-57, 2-58, 7-6, 7-7
:FORTRAN command, 1-1, 2-59, 2-60, 3-16, 7-5, 7-7, 7-16
FORTRAN compiler, 2-55-2-60, 3-2, 3-3, 4-1-4-3, 7-5
FREAD intrinsic, 6-8
FREADDIR intrinsic, 6-8
FREADLABEL intrinsic, 6-8

FREADSEEK intrinsic, 6-34
:FREERIN command, 2-61, 9-10, 9-11
FRENAME intrinsic, 6-21
FSPACE intrinsic, 6-8
FUPDATE intrinsic, 6-8
FWRITE intrinsic, 6-8, 6-38, 6-39
FWRITEDIR intrinsic, 6-8
FWRITELABEL intrinsic, 6-8
FUNLOCK intrinsic, 6-79

G

:GETRIN command, 2-61, 9-10, 9-11
Global area of stack, B-5, B-6
Global RINS, 2-61, 2-10, 2-11
Groups
 home, 3-7, 3-8
 log-on, 3-7, 6-19
 names for, 2-66, 2-70, 3-7, 4-5, 6-19
 public, 6-77
 security restrictions on, 6-77

H

Header records, 6-51, 8-11
=HEADOFF console command, 4-5, 8-10
=HEADON console command, 4-5, 8-10
:HELLO command, 1-1, 2-65, 3-1, 3-2, 3-4-3-14, 3-22
HIPRI priority, 2-67, 2-71
Home groups, 3-7, 3-8
HP 2780/3780 Emulator, 2-107

I

Implied :FILE commands, 6-53, 7-16-7-19
Input devicefiles, reporting status of, 2-131-2-133
Input/output devices,
 preparing for use of, 3-5, 3-6, 3-24-3-31
 reporting status of, 2-129, 2-130
Input/output efficiency, 6-33
Input/output sets, 6-23
Interpreter
 BASIC, 2-9, 3-20, 7-14, 7-15
 Command, 1-2, 1-5, 2-35, 3-1-3-4, 3-23, 4-2
Interrupting command execution, 3-15-3-18
Intrinsics, 1-5
IOWAIT intrinsic, 6-40

J

:JOB command, 1-1, 2-69-2-73, 4-2-4-9, 4-19, 4-20
Job fence, 3-12, 4-7, 5-1
Job listing header page, 4-2, 4-5
Job listing trailer page, 4-2, 4-10

Job name, 2-69, 4-6
Job number, 4-2, 4-5
 =JOBFENCE console command, 4-7, 5-1
 :JOBPRI System Supervisor command, 4-7
Jobs
 abnormal termination of, 4-19-4-21
 comments in, 4-16
 errors in, 2-31, 4-10, 4-21, 10-2
 example of, 4-1-4-3
 execution priorities for, 4-6, 4-7
 extra copies of output, 4-9
 identities for, 4-4
 initiation of, 2-69-2-73, 4-1-4-9
 initiation of during session, 2-153, 2-154, 4-1, 4-4,
 4-12-4-16
 input priority for, 4-6, 4-7
 listing header page
 listing trailer page, 4-2, 4-10
 names for, 2-69, 4-6
 output device for, 4-9
 output priority for, 2-72, 4-8, 4-9
 reporting status of, 2-135, 2-136
 re-startable, 2-153, 2-154
 spooling of, 2-153, 2-154
 termination of, 2-43, 4-2, 4-4, 4-10, 4-11, 4-19-4-21

K

Keyword parameters, 1-4

L

Labels, file, 6-5, 6-17, 6-57, 6-60
 =LIMIT console command, 5-1
Line printer
 header pages, 6-51, 8-11
 special forms for, 6-53, 8-11
 trailer pages, 6-51, 8-11
List files, 2-11, 2-23, 2-25, 2-27, 2-37, 2-55, 2-57, 2-59,
 2-107, 2-109, 2-111, 2-113, 2-145, 2-147, 2-149
 :LISTF command, 1-1, 2-75-2-77, 3-15, 3-16, 6-40, 6-55-
 6-57
Listing file descriptions, 2-75-2-77, 6-40, 6-55-6-57
LMAP listing, 2-83, 2-115, 7-12-7-14
Local area of stack, B-5, B-6
Lockwords for files, 2-17, 2-47, 2-93, 6-12, 6-18, 6-20,
 6-21, 6-60
Logical device number, 2-18, 2-51, 2-71, 2-102, 2-129,
 6-25, 8-1
Logical records
 blocking factor for, 6-27-6-32
 blocking of, 2-17, 2-48, 6-27-6-32
 constraints on size of, 6-31
 default sizes for, 6-32
 definition of, 6-1
 fixed-length, 6-27, 6-28
 relation to files and blocks, 6-10, 6-31
 size of, 2-17, 2-48, 6-27-6-32

 undefined length, 6-30, 6-31
 variable length, 6-27, 6-29
Logging-off, 2-21, 3-4, 3-14, 3-22
Logging-on, 1-1, 2-65, 3-1, 3-2, 3-4-3-14, 3-22
 =LOGOFF console command, 3-22, 4-19
Log-on errors, 3-13-3-14
Log-on groups, 3-7, 6-19

M

Magnetic tape files
 default record size for, 6-32
 for :STORE command, 6-66-6-68
 submitting jobs and data from, 6-34
 with unknown contents, 6-33
Master files, 2-23, 2-25, 2-27, 2-55, 2-57, 2-59, 2-109, 2-111,
 2-113, 2-145, 2-147, 2-149
Messages
 Command Interpreter Error, 10-1-10-14
 Command Interpreter Warning, 10-1, 10-15, 10-16
 Console, 10-1
 Operator, 2-127, 2-157, 9-1, 9-4, 10-1, 10-18
 Run-time, 10-1
 Segmenter Error, 10-1
 System, 10-1, 10-17
 System Error/Failure, 10-1
 User, 2-127, 2-155, 10-1, 10-18
Multi-access mode, 6-45-6-48
Multi-record mode, 2-50, 6-8, 6-36

N

Names in commands, 1-2, 1-3
New files, 2-45, 2-47, 6-5, 6-6, 6-11, 6-24, 6-55
 \$NEWPASS, 2-47, 6-21, 6-23, 7-4-7-7, 7-11
Non-breakable commands, 3-16
Non-program commands, 3-15, 3-16
NOWAIT input/output, 2-53, 6-40
 \$NULL, 6-21, 6-22, 6-23

O

Octal numbers in commands, 1-1
Old files, 2-45, 2-47, 6-7, 6-11, 6-24
 \$OLDPASS, 2-47, 3-4, 3-5, 4-2, 6-21, 6-23, 6-63, 7-4-7-8,
 7-11
OPENED devicefile state, 3-21, 3-22, 4-17-4-19
Operator, console, 9-1, 9-2, 9-4
Outfence (output fence), 4-9, 6-26
 =OUTFENCE console command, 4-17
Output devicefiles, reporting status of, 2-137-2-140
Output priority for files, 2-52
Output set, 6-23

P

P register, B-3, B-4
 Parameter lists, 1-2, 1-3
 Parameters
 keyword, 1-4
 omitted, 1-4
 positional, 1-3, 1-4
 Parity-checking, 3-24
 Passing files, 6-21, 7-4
 Passwords, 2-23, 2-65, 2-66, 2-69, 2-70, 3-8, 3-9, 4-5, 4-6
 PB register, B-3, B-4
 PCB, B-1, B-3
 PCBX area, 2-85, 2-116, B-3, B-6
 Permanent files, 2-20, 2-75-2-77, 2-89, 2-91, 2-93, 2-95,
 2-119, 2-120, 3-15, 6-11, 6-12, 6-55, 6-66, 7-5, 7-7, 9-6
 PL register, B-3, B-4
 PMAP Listing, 2-79, 2-83, 7-9, 7-10
 Positional parameters, 1-3, 1-4
 Power failures, 3-22, 4-8, 4-20
 :PREP command, 1-1, 2-79-2-81, 7-3, 7-4, 7-7, 7-9, 7-10,
 7-19
 Preparing programs, 2-79-2-86, 7-1-7-3, 7-6-7-11
 :PREPRUN command, 2-83-2-86, 7-3, 7-4, 7-11, 7-12, 7-19
 Privileged mode, 2-83, 2-115, 2-117, 7-4
 Procedure libraries, 2-84, 2-116
 Process control block, B-1, B-3
 Process Control Block Extension area of stack, 2-85, 2-116,
 B-3, B-6
 Processes, B-1-B-9
 Program base register, B-3, B-4
 Program counter register, B-3, B-4
 Program entry-point, 2-83, 2-115, 2-117, 7-12
 Program files, 2-27, 2-28, 2-57, 2-58, 2-79, 2-113, 2-114,
 2-115-2-117, 7-3, 7-7, 7-9
 Program limit register, B-3, B-4
 Program units, 7-1
 Programmatic execution of commands, 1-5
 Programs
 aborting of, 2-5, 3-18, 3-19
 allocation of, 7-1, 7-3, B-1, B-3
 comments in, 2-29
 compilation of, 7-1-7-9
 execution of, 2-83-2-86, 2-115-2-117, 7-1, 7-3, 7-4,
 7-8-7-13, B-1, B-9
 listing of loaded, 7-12-7-14
 listing of prepared, 2-79, 2-83, 7-9, 7-10
 preparation of, 2-79-2-86, 7-1-7-3, 7-6-7-11
 Prompt character for MPE commands, 1-2, 1-3
 :PTAPE command, 2-87, 2-89, 9-6-9-10
 :PURGE command, 1-1, 3-15, 6-62, 6-63

Q

Q register, B-8, B-9
 Quiet mode, 5-5, 9-3, 9-4

R

RBMs, 2-81, 7-1-7-3, 7-10, 7-15
 Reading, anticipatory, 6-34
 Reading data from standard input file/device, 3-19
 READY devicefile state, 3-21, 3-22, 4-17-4-19
 Ready List, 3-11
 Record (line) terminators, 3-24
 Records
 blocking factor for, 6-27-6-32
 blocking of, 2-17, 2-48, 6-27-6-32
 constraints on size of, 6-31
 default size for, 6-32
 definition of, 6-1
 fixed-length, 6-27, 6-28
 relation to files and blocks, 6-10, 6-31
 size of, 2-17, 2-48, 6-27-6-32
 undefined-length, 6-30, 6-31
 variable-length, 6-27, 6-29
 Reference specifications, 1-1, 1-2, 2-1-2-157
 -REFUSE console command, 4-14
 Register
 P, B-3, B-4
 PB, B-3, B-4
 PL, B-3, B-4
 program base, B-3, B-4
 program counter, B-3, B-4
 program limit, B-3, B-4
 Q, B-8, B-9
 S, B-8, B-9
 Z, B-5, B-6
 :RELEASE command, 1-1, 2-91, 6-61, 6-80
 Relocatable binary modules, 2-81, 7-1-7-3, 7-10, 7-15
 Relocatable library files, 2-80, 2-85
 :RENAME command, 2-93, 6-21, 6-60, 6-61, 6-80, 6-81
 :REPORT command, 2-95, 3-14, 5-6-5-8, 6-42
 Report Program Generator (RPG) compiler, 2-109, 2-114
 :RESET command, 2-97, 6-54
 :RESETDUMP command, 2-99, 2-125
 Re-setting formal file designator, 2-97
 Resource accounting and limits, 5-6-5-8
 Re-startable jobs, 3-22, 4-8, 4-20
 :RESTORE command, 1-1, 2-101-2-103, 6-64, 6-66-6-74,
 7-19
 :RESUME command, 2-105, 3-16-3-18
 =RESUMEJOB console command, 5-1
 Resuming program execution, 2-105
 RINs, 2-61, 2-63, 9-10, 9-11
 :RJE command, 2-107, 7-15
 RL files, 2-80, 2-85
 :RPG command, 1-1, 2-109, 2-110, 7-6
 RPG compiler, 2-109, 2-114, 7-6
 :RPGGO command, 2-111, 7-8
 :RPGPREP command, 2-113, 2-114, 7-6, 7-7
 :RUN command, 1-1, 2-31, 2-54, 2-115-2-117, 3-16, 7-4,
 7-9, 7-12, 7-19
 Running programs, 2-83-2-86, 2-115-2-117, 3-16, 7-4, 7-9,
 7-12, 7-19

S

S register, B-8, B-9
:SAVE command, 1-1, 2-119, 2-120, 3-4, 4-2, 4-3, 6-11, 6-63, 6-64, 7-4, 7-6, 7-8, 7-9
SAVE-FAST files, BASIC, 2-11-2-16, 7-15
:SECURE command, 1-1, 2-91, 2-121, 6-62, 6-81
Security provisions of MPE, 2-7, 3-8, 2-91, 2-121, 3-8, 6-12, 6-37, 6-61, 6-62, 6-74-6-81
:SEGMENTER command, 2-123, 7-15
Segmenter subsystem, 2-123, 3-3, 4-2, 7-3
Segments
 code, B-1, B-2
 data, B-1, B-2
Sequential-accessing of files, 6-8, 6-9
Session name, 2-65, 3-10
Session/job execution priority class, 2-67, 2-71, 3-11, 3-12, 4-7
Session/job input priority, 2-67, 2-71, 3-12
Session/job number, 2-68, 3-1, 3-2
Session/job processing states, 5-1-5-6
Session/job status, 5-1-5-6
Session/job temporary file domain, 6-11, 6-12, 6-24
Session/job Temporary File Directory, 6-5, 6-7, 6-11, 6-12, 6-24
Session/job temporary files, 6-5, 6-7, 6-11, 6-12, 6-24
Sessions
 abnormal termination of, 3-22
 errors in, 3-23, 10-2
 example of, 3-1-3-4
 initiation of, 1-1, 2-65-2-68, 3-1, 3-2, 3-4-3-14, 3-22
 names of, 2-65, 3-10
 reporting status of, 2-135, 2-136
 scheduling of, 3-10-3-12
 termination of, 2-21, 3-4, 3-13, 3-14, 3-22
:SETDUMP command, 2-99, 2-125
:SETMSG command, 2-127, 2-156, 9-3, 9-4
:SHOWDEV command, 1-1, 2-129, 2-130, 5-6, 8-1-8-3
:SHOWIN command, 1-1, 2-131-2-133, 3-21, 5-6, 8-8-8-11
:SHOWJOB command, 1-1, 2-135, 2-136, 3-10, 3-15, 4-12, 5-1-5-6, 9-1
:SHOWOUT command, 1-1, 2-137-2-140, 3-21, 4-19, 8-13-8-15
:SHOWTIME command, 2-141, 9-12
=SHUTDOWN console command, 3-22, 4-19
Source text files, 2-23, 2-25, 2-27, 2-55, 2-57, 2-59, 2-109, 2-111, 2-113, 2-145, 2-147, 2-149
Space limit for files, 3-13, 3-14, 5-6-5-8
Special keyboard functions, 3-22
:SPEED command, 2-143, 9-5
:SPL command, 1-1, 2-145, 2-146, 3-16, 7-5
SPL compiler, 2-145-2-150, 3-16
:SPLGO command, 2-147, 7-8
:SPLPREP command, 2-149, 7-6
=SPOOL console command, 3-22, 4-18, 4-21
Spooled devicefiles, 3-20, 3-21, 3-22, 4-17-4-19, 6-3, 6-10, 8-4
Spooling, 3-20, 3-21, 3-22, 4-4, 4-9, 4-17-4-19, 8-4, 8-6, 8-7, 8-12, 2-153, 2-154, 6-81-6-83
Stack, B-1-B-9
Stack Marker, B-8
Stack Overflow area, B-5, B-7

Stackdump facility, 2-99, 2-125
Standard input device/file, 3-1, 3-19, 3-20, 4-2, 4-11, 4-12, 6-22, 6-23, 7-4, 7-6, 7-7, 7-8
Standard listing device/file, 3-1, 4-2, 6-22, 6-23, 7-4, 7-7, 7-8
\$STDIN, 3-19, 6-22, 6-23, 6-24
\$STDINX, 3-19, 4-11, 6-22, 6-23, 6-24
\$STDLIST, 6-22, 6-23
:STORE command, 1-1, 2-151, 2-152, 3-6, 6-64-6-70, 7-19
:STREAM command, 2-153, 2-154, 4-4, 4-12-4-16
Streaming jobs, 4-1, 4-2, 4-4, 4-12-4-16
=STREAMS console command, 4-12
Subsystem break function, 3-24
:SYSDUMP command, 2-102, 6-66, 6-71, 6-74
System BREAK function, 2-1, 3-7, 3-15-3-19, 3-24, 4-15
System File Directory, 6-5, 6-11, 6-12
System file domain, 6-11, 6-12, 6-24

T

Tape mode, 3-24, 9-6-9-10
:TELL command, 1-1, 2-127, 2-155, 2-156, 3-10, 3-15, 5-5, 9-1-9-4
=TELL console command, 2-127, 3-10, 5-5, 9-4
:TELLOP command, 1-1, 2-157, 3-10, 3-15, 9-4
Terminals
 block-mode for, 3-29-3-31
 character mode for, 3-29
 correcting characters on, 3-28
 MPE-supported, A-1
 operating, 3-4, 3-5, 3-8, 3-9, 3-24-3-31
 reading paper tapes at, 9-6-9-10
 speed of, 2-143, 3-24, 9-5
 types of, 2-66, 2-70, 3-9, 3-28-3-31, 4-5
Terminating jobs, 2-43, 4-2, 4-4, 4-10, 4-11, 4-19
Time, display of, 2-141
Time limits on central processor use, 2-67, 2-71, 2-95, 3-1, 3-4, 3-10, 3-13, 3-14, 3-22, 4-1, 4-6, 4-7, 4-10, 5-6-5-10
Top-of-stack, B-3, B-7, B-8
Trailer records, 6-5, 8-11
Transmission (duplex) mode, 3-8, 3-9, 3-25, 3-26

U

Undefined-length records, 6-30, 6-31
Unused data area, B-5, B-6
User name, 2-65, 2-69, 3-5, 4-4
User subprogram library files, 2-11, 2-23, 2-55, 2-58, 2-59, 2-79, 2-81, 2-83, 2-85, 2-86, 2-87, 2-109, 2-110, 2-111, 2-114, 2-145-2-150, 7-1-7-3, 7-5, 7-6, 7-7, 7-9, 7-11
User/System Dialogue, 2-4
Users, communicating with, 9-1-9-4
USL files, 2-11, 2-23, 2-55, 2-58, 2-59, 2-79, 2-81, 2-83, 2-85, 2-86, 2-87, 2-109, 2-110, 2-111, 2-114, 2-145-2-150, 7-1-7-3, 7-5, 7-6, 7-7, 7-9, 7-11
Utility operations, 9-1-9-12

V

Variable-length records, 6-27, 6-29

W

=WARN console command, 9-4
Working stack, B-5, B-6

X

X-OFF control character, 2-87, 9-6-9-11

Z

Z-register, B-5, B-6
Z-DL area, 2-79, 2-84, 2-116, B-5, B-6
Z-Q (initial) area, 2-79, 2-84, 2-116, B-5, B-6

Part No. 30000-90009
Printed in U.S.A. 6/76

HEWLETT  PACKARD

Sales and service from 172 offices in 65 countries.
5303 Stevens Creek Blvd., Santa Clara, California 95050