

HP 32000C
MPE/3000 Operating System
reference manual



HEWLETT  PACKARD

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

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 | Pages | Effective Date |
|----------------------|-----------------------|----------------------|-----------------------|
| Title | Jun 1976 | 6-54 | Mar 1976 |
| ii to v | Mar 1976 | 6-55 | Mar 1976 |
| vi to xv | Jan 1975 | 6-56 | Jan 1975 |
| xvi to xvii | Oct 1975 | 6-57 | Oct 1975 |
| xviii | Jan 1975 | 6-58 to 6-63 | Jan 1975 |
| xix | Mar 1976 | 7-1 to 7-2 | Jan 1975 |
| xx | Jan 1975 | 7-3 | Oct 1975 |
| xxi | Mar 1976 | 7-4 to 7-8 | Jan 1975 |
| Part I Divider | Jan 1975 | 7-9 | Oct 1975 |
| 1-1 to 1-10 | Jan 1975 | 7-10 to 7-30 | Jan 1975 |
| 2-1 to 2-29 | Jan 1975 | 8-1 | Oct 1975 |
| Part 2 Divider | Jan 1975 | 8-2 | Jan 1975 |
| 3-1 to 3-11 | Jan 1975 | 8-3 to 8-4 | Oct 1975 |
| 3-12 to 3-14 | Oct 1975 | 8-5 | Jan 1975 |
| 3-15 to 3-19 | Jan 1975 | 8-6 to 8-7 | Oct 1975 |
| 3-20 | Oct 1975 | 8-8 to 8-12 | Jan 1975 |
| 3-21 to 3-28 | Jan 1975 | 8-13 to 8-13A | Oct 1975 |
| 4-1 to 4-9 | Jan 1975 | 8-14 to 8-15 | Jan 1975 |
| 4-10 | Oct 1975 | 8-16 to 8-17 | Jun 1976 |
| 4-11 to 4-27 | Jan 1975 | 8-18 | Oct 1975 |
| 5-1 to 5-14 | Jan 1975 | 8-18A | Mar 1976 |
| 5-15 | Oct 1975 | 8-19 to 8-31 | Jan 1975 |
| 5-16 to 5-34 | Jan 1975 | 8-32 | Oct 1975 |
| 5-35 to 5-36 | Oct 1975 | 8-33 to 8-44 | Jan 1975 |
| 5-37 to 5-38 | Jan 1975 | 8-45 | Oct 1975 |
| 5-39 to 5-41 | Oct 1975 | 8-46 to 8-59 | Jan 1975 |
| 5-42 to 5-55 | Jan 1975 | 8-60 | Oct 1975 |
| 6-1 to 6-17 | Jan 1975 | 8-61 to 8-63 | Jan 1975 |
| 6-18 | Oct 1975 | 8-64 to 8-65 | Oct 1975 |
| 6-19 to 6-21 | Jan 1975 | 8-66 to 8-86 | Jan 1975 |
| 6-22 | Oct 1975 | 8-87 | Oct 1975 |
| 6-23 to 6-37 | Jan 1975 | 8-88 to 8-98 | Jan 1975 |
| 6-38 | Oct 1975 | 9-1 to 9-11 | Jan 1975 |
| 6-39 | Jan 1975 | 10-1 | Oct 1975 |
| 6-40 | Oct 1975 | 10-2 to 10-13 | Jan 1975 |
| 6-41 to 6-43 | Jan 1975 | 10-14 | Oct 1975 |
| 6-44 | Oct 1975 | 10-15 to 10-16 | Jan 1975 |
| 6-45 | Mar 1976 | 10-17 to 10-26 | Mar 1976 |
| 6-46 | Jan 1975 | 10-27 to 10-31 | Jan 1975 |
| 6-47 | Mar 1976 | 10-32 to 10-36 | Oct 1975 |
| 6-48 to 6-53 | Jan 1975 | | |

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

List of Effective Pages (continued)

| Pages | Effective Date | Pages | Effective Date |
|----------------------|----------------|----------------------|----------------|
| Part 3 Divider | Jan 1975 | B-1 to B-10 | Jan 1975 |
| 11-1 to 11-6 | Jan 1975 | C-1 to C-9 | Jan 1975 |
| 11-7 | Jun 1976 | C-10 to C-10.2 | Oct 1975 |
| 11-8 to 11-13 | Jan 1975 | C-11 to C-12 | Jan 1975 |
| 11-14 | Jun 1976 | C-12.1 | Oct 1975 |
| 11-15 to 11-29 | Jan 1975 | C-13 to C-58 | Jan 1975 |
| 12-1 to 12-7 | Jan 1975 | C-58.1 | Mar 1976 |
| 13-1 to 13-2 | Jan 1975 | C-59 to C-60 | Jan 1975 |
| 14-1 to 14-5 | Jan 1975 | C-61 | Oct 1975 |
| 14-6 | Oct 1975 | C-62 to C-88 | Jan 1975 |
| 14-7 to 14-8 | Jan 1975 | D-1 | Jan 1975 |
| 14-9 | Jun 1976 | D-2 | Oct 1975 |
| 14-10 to 14-13 | Jan 1975 | E-1 to E-2 | Jan 1975 |
| Part 4 Divider | Jan 1975 | F-1 to F-2 | Jan 1975 |
| A-1 to A-5 | Jan 1975 | G-1 | Jan 1975 |

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 | Jan 1975 |
| Update Package #1 | Oct 1975 |
| Update Package #2 | Mar 1976 |
| Update Package #3 | Jun 1976 |



Preface

This manual explains how to compile and execute programs, manipulate files, request utility functions, and perform other standard programming operations under the Multiprogramming Executive Operating System (MPE/3000, Release C) for the HP 3000 Computer. In addition, it shows how to use the various optional capabilities of the system. Subjects are arranged in functional order, beginning with the simplest, most commonly-used operations and proceeding toward more difficult, sophisticated functions. Once the user has familiarized himself with this book, he can use the extensive index in the back to quickly and easily find any information he needs.

This manual is a reference book rather than a tutorial text for new programmers. The reader should understand the fundamental techniques of programming and the operating principles of the HP 3000 Computer. He should also examine the following manual for an overview of the inter-relationships between the main hardware and software features offered:

HP 3000 Software General Information Manual

The user may also want to read the following manuals for additional information, depending upon the applications he plans:

HP 3000 Systems Programming Language

HP 3000 Systems Programming Language Textbook

MPE/3000 Operating System, Console Operator's Guide

MPE/3000 Operating System, System Manager/System Supervisor Manual



Guide for Readers

The following guide shows the sections of the manual that should be read to perform particular tasks:

| To: | Read Section: | Prerequisite Sections: |
|-----------------------------------------------------------------|---------------|------------------------|
| Initiate and terminate batch jobs or interactive sessions. | III | None |
| Compile, prepare, and execute programs. | IV | III |
| Call MPE/3000 subsystems. | IV | III |
| Create and manage files through MPE/3000 commands. | V | III, IV |
| Read, write, and update files. | VI | III, IV |
| Manipulate files and obtain file-access and error information. | VI | III, IV |
| Alter user subprogram libraries. | VII | III, V |
| Maintain external library procedures. | VII | III, V |
| List system information. | VIII | III |
| Request binary/ASCII code conversion. | VIII | III |
| Return system information to the user's program. | VIII | III |
| Enable or disable traps. | VIII | III |
| Manage interprocess communication through the job control word. | VIII | III |
| Reserve and allocate user resources. | IX | III |

| To: | Read Section: | Prerequisite Sections: |
|----------------------------------------------------------------|---------------|------------------------|
| Diagnose errors. | X | None |
| Handle processes. (Optional Capability.) | XI | II, III, IV |
| Manage Data Segments. (Optional Capability.) | XII | II, III, IV |
| Manage multiple RIN's. (Optional Capability.) | XIII | II, III, IV, IX |
| Execute programs in privileged mode. (Optional Capability.) | XIV | II, III, IV |

CONTENTS

| | |
|---------------------------------------------------|-----|
| Preface | iii |
| Guide for Readers | v |
| | |
| PART 1 System Overview | |
| | |
| SECTION I Introduction to MPE/3000 | 1-1 |
| FEATURES | 1-1 |
| Multiprogramming | 1-1 |
| General-Purpose Versatility | 1-1 |
| Choice of Programming Languages | 1-2 |
| Operating Simplicity | 1-2 |
| Input/Output Conveniences | 1-3 |
| Processing Efficiency | 1-3 |
| Accounting Facility | 1-5 |
| Logging Facility | 1-5 |
| System-to-System Compatibility | 1-5 |
| Program and File Security | 1-5 |
| System Manager and System Supervisor Capabilities | 1-5 |
| SOFTWARE | 1-6 |
| HARDWARE | 1-7 |
| | |
| SECTION II How MPE/3000 Operates | 2-1 |
| MPE/3000 RESIDENCE | 2-2 |
| PROCESSES | 2-3 |
| PROCEDURES | 2-5 |
| SYSTEM LIBRARY | 2-6 |

| | |
|------------------------------------|------|
| USER INTERFACE | 2-6 |
| Commands | 2-6 |
| Intrinsic Calls | 2-8 |
| INPUT/OUTPUT | 2-8 |
| File Management | 2-9 |
| Scheduling | 2-9 |
| COMPILATION | 2-11 |
| PROGRAM PREPARATION | 2-13 |
| ALLOCATION/EXECUTION | 2-13 |
| SPOOLING | 2-14 |
| PCB/CODE SEGMENT/STACK INTERACTION | 2-14 |
| MEMORY MANAGEMENT | 2-20 |
| Main-Memory Linkages | 2-20 |
| Main-Memory Use | 2-21 |
| USER JOB PROCESSING | 2-21 |
| Batch Programs (Jobs) | 2-21 |
| Interactive Programs (Sessions) | 2-23 |
| ACCOUNT/GROUP/USER ORGANIZATION | 2-24 |
| CAPABILITY SETS | 2-25 |
| User Attributes | 2-27 |
| File Access Attributes | 2-28 |
| Capability-Class Attributes | 2-28 |
| Local Attributes | 2-29 |
| Program Capability Sets | 2-29 |

PART 2 Standard Capabilities

| | |
|------------------------------------------------|------------|
| SECTION III Communicating with MPE/3000 | 3-1 |
| COMMANDS | 3-1 |
| Positional Parameters | 3-2 |
| Keyword Parameters | 3-3 |
| Continuation Characters | 3-4 |
| Command Description Format | 3-4 |
| Command Errors | 3-6 |
| INTRINSIC CALLS | 3-6 |
| Intrinsic Description Format | 3-8 |
| Intrinsic Call Errors | 3-9 |

| | |
|------------------------------------------------------------------------------|------|
| BATCH JOBS | 3-10 |
| Initiating Batch Jobs | 3-10 |
| Terminating Batch Jobs | 3-16 |
| Typical Job Structures | 3-16 |
| INTERACTIVE SESSIONS | 3-19 |
| Initiating Sessions | 3-19 |
| Interrupting Program Execution Within Sessions | 3-22 |
| Terminating Sessions | 3-23 |
| Typical Session Structure | 3-24 |
| READING DATA FROM OUTSIDE STANDARD INPUT STREAM | 3-25 |
| PREMATURE JOB OR SESSION TERMINATION | 3-26 |
| INTRODUCING JOBS FOR SCHEDULING | 3-28 |
| | |
| SECTION IV Compiling, Interpreting, Preparing, and Executing Programs | 4-1 |
| REFERENCING FILES | 4-1 |
| Specifying Files as Command Parameters | 4-1 |
| Specifying Files by Default | 4-4 |
| USING THE BASIC/3000 INTERPRETER | 4-5 |
| COMPILING/PREPARING/EXECUTING PROGRAMS | 4-6 |
| Compilation Only | 4-7 |
| Compilation/Preparation | 4-9 |
| Compilation/Preparation/Execution | 4-11 |
| Preparation Only | 4-12 |
| Preparation/Execution | 4-17 |
| Execution | 4-21 |
| CALLING MPE/3000 SUBSYSTEMS | 4-22 |
| EDIT/3000 | 4-22 |
| STAR/3000 | 4-23 |
| TRACE/3000 | 4-23 |
| SORT/3000 | 4-23 |
| SDM/3000 | 4-24 |
| MPE/3000 Segmenter | 4-24 |
| 2780/3780 Emulator | 4-25 |
| SAMPLE PROGRAMS | 4-26 |
| | |
| SECTION V Managing Files | 5-1 |
| FILE CHARACTERISTICS | 5-1 |
| FILE/DEVICE RELATIONSHIPS | 5-3 |
| NON-SHARABLE DEVICE ACCESS | 5-3 |
| FILE DOMAINS | 5-3 |

| | |
|--------------------------------------------------|------|
| FILE LABELS | 5-4 |
| FILE ACCESSING | 5-4 |
| System-Defined Files | 5-5 |
| User Pre-Defined Files | 5-5 |
| New Files | 5-6 |
| Old Files | 5-6 |
| Filereference Formats | 5-7 |
| Lockwords | 5-8 |
| File System Accounting | 5-9 |
| Input/Output Sets | 5-10 |
| SPECIFYING FILE CHARACTERISTICS | 5-10 |
| Command Parameters | 5-14 |
| Accessing Files Already in Use | 5-19 |
| Re-Specifying File Names | 5-21 |
| Passing Files | 5-22 |
| Issuing Detailed File Specifications | 5-23 |
| Implicit File Commands | 5-25 |
| Command File Parameters | 5-26 |
| RESETTING A FORMAL FILE DESIGNATOR | 5-29 |
| CREATING A NEW FILE | 5-30 |
| SAVING A FILE | 5-31 |
| DELETING A FILE | 5-32 |
| LISTING FILE SETS | 5-32 |
| DUMPING FILES OFF-LINE | 5-35 |
| RETRIEVING DUMPED FILES | 5-40 |
| RENAMING A FILE | 5-44 |
| SPECIFYING FILE SECURITY | 5-45 |
| Account-Level Security | 5-47 |
| Group-Level Security | 5-48 |
| File-Level Security | 5-49 |
| Changing File-Level Provisions | 5-51 |
| Suspending Security Provisions | 5-53 |
| LOCKING FILES | 5-54 |
| FILE MANAGEMENT COMMAND FILE-TYPE SUMMARY | 5-55 |

| | | |
|--------------------|-----------------------------------------------------------|-------------|
| SECTION VI | Accessing and Altering Files | 6-1 |
| | OPENING FILES | 6-3 |
| | Files On Non-Sharable Devices | 6-4 |
| | Devicefile Management | 6-5 |
| | Record Formats | 6-15 |
| | Foptions Parameter | 6-20 |
| | Aoptions Parameter | 6-25 |
| | CLOSING FILES | 6-30 |
| | READING SEQUENTIAL FILES | 6-32 |
| | READING DIRECT-ACCESS FILES | 6-34 |
| | OPTIMIZING DIRECT-ACCESS FILE-READING | 6-36 |
| | WRITING ON SEQUENTIAL FILES | 6-37 |
| | WRITING ON DIRECT-ACCESS FILES | 6-41 |
| | READING LABELS | 6-43 |
| | WRITING LABELS | 6-44 |
| | UPDATING FILES | 6-45 |
| | SPACING ON SEQUENTIAL FILES | 6-46 |
| | RESETTING LOGICAL RECORD POINTER | 6-47 |
| | OBTAINING FILE ACCESS INFORMATION | 6-48 |
| | OBTAINING FILE-ERROR INFORMATION | 6-52 |
| | DIRECTING FILE CONTROL OPERATIONS | 6-56 |
| | DECLARING ACCESS-MODE OPTIONS | 6-58 |
| | LOCKING AND UNLOCKING FILES | 6-60 |
| | RENAMING A FILE | 6-60 |
| | DETERMINING INTERACTIVE AND DUPLICATIVE FILE PAIRS | 6-61 |
| | FILE INTRINSIC FILE-TYPE SUMMARY | 6-63 |
| | | |
| SECTION VII | Managing Program Libraries | 7-1 |
| | MANAGING USER SUBPROGRAM LIBRARIES (USL's) | 7-1 |
| | Creating New USL's | 7-3 |
| | Designating USL's For Management By The User | 7-3 |
| | Activating Entry Points | 7-5 |
| | Deactivating Entry points | 7-6 |
| | Deleting RBM's | 7-7 |
| | Assigning New Segment Names to RBM's | 7-8 |
| | Transferring RBM's | 7-9 |
| | Listing RBM's | 7-10 |
| | Preparing Program Files | 7-12 |



| | |
|-------------------------------------------------------|-------|
| USING EXTERNAL PROCEDURE LIBRARIES | 7-15 |
| Relocatable Libraries | 7-15 |
| Segmented Libraries | 7-16 |
| CREATING AND MAINTAINING RELOCATABLE LIBRARIES (RL's) | 7-18 |
| Creating a Relocatable Procedure Library (RL) File | 7-18 |
| Designating RL's for Management by the User | 7-19 |
| Adding a Procedure to an RL | 7-19 |
| Deleting an Entry-Point or Procedure from an RL | 7-19 |
| Listing Procedures in an RL | 7-20 |
| Examples of Relocatable Library Management Commands | 7-22 |
| CREATING AND MAINTAINING SEGMENTED LIBRARIES (SL's) | 7-23 |
| Creating a Segmented Procedure Library (SL) File | 7-23 |
| Designating SL's for Management by the User | 7-23 |
| Adding a Procedure to an SL | 7-24 |
| Deleting an Entry-Point or Segment From an SL | 7-24 |
| Listing Procedures in an SL | 7-24 |
| SETTING RBM INTERNAL FLAGS | 7-27 |
| DYNAMIC LOADING OF LIBRARY PROCEDURES | 7-27 |
| Dynamic Loading | 7-28 |
| Dynamic Unloading | 7-29 |
| | |
| SECTION VIII Requesting Utility Operations | 8-1 |
| LISTING DATE, TIME, AND ACCOUNTING INFORMATION | 8-1 |
| Date and Time | 8-2 |
| Accounting Charges | 8-2 |
| DETERMINING JOB STATUS | 8-2 |
| DETERMINING DEVICE STATUS | 8-4 |
| OBTAINING DEVICEFILE INFORMATION | 8-5 |
| TRANSMITTING MESSAGES | 8-8 |
| INSERTING COMMENTS IN COMMAND STREAM | 8-9 |
| REQUESTING ASCII/BINARY NUMBER CONVERSION | 8-10 |
| Converting Numbers from ASCII to Binary Code | 8-10 |
| Converting Numbers from Binary to ASCII Code | 8-11 |
| REQUESTING CHARACTER TRANSLATION | 8-13 |
| TRANSMITTING PROGRAM INPUT/OUTPUT | 8-13A |
| Reading Input | 8-13A |
| Writing Output to the Listing Device | 8-15 |
| Writing Output to the Operator's Console | 8-16 |

| | |
|----------------------------------------------------------------------|------|
| OBTAINING SYSTEM TIMER INFORMATION | 8-17 |
| System Timer Bit Count | 8-17 |
| Current Time | 8-18 |
| DETERMINING THE USER'S ACCESS MODE AND ATTRIBUTES | 8-19 |
| OBTAINING PROCESS RUN-TIME (USE OF THE CENTRAL PROCESSOR) | 8-22 |
| SEARCHING ARRAYS | 8-22 |
| FORMATTING COMMAND PARAMETERS | 8-25 |
| EXECUTING MPE/3000 COMMANDS PROGRAMMATICALLY | 8-28 |
| ENABLING AND DISABLING TRAPS | 8-30 |
| Arithmetic Trap | 8-31 |
| Library Trap | 8-33 |
| System Trap | 8-34 |
| CONTROL-Y Traps | 8-35 |
| Trap Procedure Execution | 8-36 |
| CHANGING STACK SIZES | 8-44 |
| Changing the DL/DB Area Size | 8-44 |
| Changing the Z-DB Area Size | 8-45 |
| REQUESTING A PROCESS BREAK | 8-46 |
| TERMINATING A PROCESS | 8-47 |
| Termination | 8-47 |
| Abort | 8-47 |
| SETTING BREAKPOINTS AND DISPLAYING/MODIFYING MEMORY OR REGISTER DATA | 8-48 |
| Invoking DEBUG | 8-48 |
| DEBUG Command Format | 8-50 |
| Setting Breakpoints | 8-51 |
| Clearing Breakpoints | 8-52 |
| Resuming Program Execution | 8-53 |
| Switching Display Output to a File | 8-54 |
| Displaying Register Contents | 8-54 |
| Displaying Memory Contents | 8-55 |
| Modifying Register Contents | 8-56 |
| Modifying Memory Contents | 8-57 |
| Requesting Trace of Stack Markers | 8-58 |
| Calculating an Expression | 8-59 |
| DUMPING THE STACK | 8-59 |
| Callable Stack Dump | 8-59 |
| Abort Stack Analysis | 8-63 |
| INTERPROCESS COMMUNICATION | 8-67 |
| VERIFYING DIAGNOSTIC DEVICE ASSIGNMENT | 8-70 |

| | |
|----------------------------------------------------------------------|-------|
| INTRINSICS FOR COMPILER WRITERS | 8-71 |
| Initializing Buffers for USL Files | 8-71 |
| Changing the Directory Block/Information Block Size on a USL File | 8-72 |
| Changing the Size of USL File | 8-72 |
| USL File Intrinsic Error Numbers | 8-73 |
| CHANGING TERMINAL CHARACTERISTICS | 8-74 |
| Types of Terminals | 8-74 |
| IBM 2741 Communication Terminal Interface | 8-78 |
| Changing Terminal Speed | 8-78 |
| Changing Input Echo Facility | 8-81 |
| Enabling and Disabling System Break Function | 8-83 |
| Enabling and Disabling Subsystem Break Function | 8-84 |
| Enabling and Disabling Parity-Checking | 8-85 |
| Enabling and Disabling Tape-Mode Option | 8-86 |
| Reading Paper Tapes Without X-OFF Control | 8-87 |
| Enabling and Disabling the Terminal Input Timer | 8-88 |
| Reading the Terminal Input Timer | 8-89 |
| Defining Line-Termination Characters For Terminal Input | 8-90 |
| DEVICE CHARACTERISTICS | 8-91 |
| Paper Tape Reader | 8-91 |
| Paper Tape Punch | 8-94 |
| Card Reader | 8-95 |
| Line Printer | 8-95 |
| Magnetic Tape | 8-96 |
| Card Punch | 8-96 |
| Printing Reader/Punch | 8-96 |
| SECTION IX Resource Management | 9-1 |
| INTERJOB LEVEL | 9-2 |
| Acquiring Global RIN's | 9-2 |
| Locking and Unlocking Global RIN's | 9-2 |
| Freeing Global RIN's | 9-6 |
| INTER-PROCESS LEVEL | 9-6 |
| Acquiring Local RIN's | 9-6 |
| Locking and Unlocking Local RIN's | 9-7 |
| Freeing Local RIN's | 9-9 |
| LOCKING AND UNLOCKING FILES | 9-9 |
| SECTION X MPE/3000 Messages | 10-1 |
| COMMAND INTERPRETER ERROR MESSAGES | 10-2 |
| COMMAND INTERPRETER WARNING MESSAGES | 10-15 |

| | |
|--------------------------|-------|
| RUN-TIME MESSAGES | 10-17 |
| USER MESSAGES | 10-27 |
| OPERATOR MESSAGES | 10-27 |
| SYSTEM MESSAGES | 10-28 |
| FILE INFORMATION DISPLAY | 10-29 |
| SEGMENTER ERROR MESSAGES | 10-32 |

PART 3 Optional Capabilities

| | |
|----------------------------------------------------------------|-------------|
| SECTION XI Process-handling Optional Capability | 11-1 |
| PROCESS LIFE-CYCLE | 11-1 |
| PROCESS-HANDLING | 11-7 |
| Creating Processes | 11-7 |
| Activating Processes | 11-12 |
| Suspending Processes | 11-13 |
| Deleting Processes | 11-14 |
| Interprocess Communication | 11-16 |
| Testing Mailbox Status | 11-17 |
| Sending Mail | 11-18 |
| Receiving (Collecting) Mail | 11-20 |
| Avoiding Deadlocks | 11-22 |
| Rescheduling Processes | 11-22 |
| Determining Source of Activation | 11-25 |
| Determining Father Process | 11-25 |
| Determining Son Process | 11-26 |
| Determining Process Priority and State | 11-27 |
| SECTION XII Data-segment Management Optional Capability | 12-1 |
| CREATING AN EXTRA DATA SEGMENT | 12-1 |
| DELETING AN EXTRA DATA SEGMENT | 12-3 |
| TRANSFERRING DATA FROM AN EXTRA DATA SEGMENT TO STACK | 12-4 |
| TRANSFERRING DATA FROM STACK TO EXTRA DATA SEGMENT | 12-5 |
| CHANGING SIZE OF DATA SEGMENT | 12-7 |

| | | |
|---------------------|--------------------------------------------------------------------|------|
| SECTION XIII | Multiple Resource Identification Number Optional Capability | 13-1 |
|---------------------|--------------------------------------------------------------------|------|

| | | |
|--------------------|--------------------------------------------|-------|
| SECTION XIV | Privileged Mode Optional Capability | 14-1 |
| | PERMANENTLY PRIVILEGED PROGRAMS | 14-2 |
| | TEMPORARILY PRIVILEGED PROGRAMS | 14-2 |
| | ENTERING PRIVILEGED MODE | 14-3 |
| | ENTERING NON-PRIVILEGED MODE | 14-4 |
| | MOVING THE DB-POINTER | 14-4 |
| | OTHER DATA-SEGMENT INTRINSICS | 14-6 |
| | SCHEDULING PROCESSES | 14-6 |
| | PRIVILEGED MODE DEBUG EXTENSIONS | 14-10 |

PART 4 Appendices

| | | |
|------------|----------------------------|-----|
| APPENDIX A | ASCII Character Set | A-1 |
| APPENDIX B | Summary of Commands | B-1 |
| APPENDIX C | Summary of Intrinsic Calls | C-1 |
| APPENDIX D | Intrinsic Error Numbers | D-1 |
| APPENDIX E | Disc File Labels | E-1 |
| APPENDIX F | :STORE Tape Format | F-1 |
| APPENDIX G | End-of-File Indication | G-1 |

INDEX

INDEX OF COMMANDS

INDEX OF INTRINSICS

FIGURES

| | | |
|-------------|---------------------------------------|------|
| Figure 1-1. | Small Batch System | 1-8 |
| Figure 1-2. | Small Interactive System | 1-9 |
| Figure 1-3. | Combined Batch and Interactive System | 1-9 |
| Figure 1-4. | Large Processing System | 1-10 |
| Figure 2-1. | Code-Sharing and Data Privacy | 2-3 |
| Figure 2-2. | Process Organization | 2-4 |
| Figure 2-3. | User/Software/Hardware Interface | 2-7 |
| Figure 2-4. | Scheduling Queues | 2-10 |

| | | |
|---------------|-----------------------------------------------|-------|
| Figure 2-5. | Program Management | 2-12 |
| Figure 2-6. | Code Segment and Associated Registers | 2-15 |
| Figure 2-7. | Data (Stack) Segment and Associated Registers | 2-17 |
| Figure 2-8. | Stack Operation | 2-19 |
| Figure 2-9. | Batch Job History | 2-22 |
| Figure 2-10. | Account/User/Group Organization | 2-26 |
| Figure 4-1. | Listing of Prepared Program | 4-16 |
| Figure 4-2. | Listing of Loaded Program | 4-20 |
| Figure 5-1. | Actions Resulting From Multi-Access of Files | 5-20 |
| Figure 6-1. | Devicefile States | 6-11 |
| Figure 6-2. | Devicefile Access | 6-13 |
| Figure 6-3. | Foptions Bit Summary | 6-24 |
| Figure 6-4. | Aoptions Bit Summary | 6-28 |
| Figure 6-5. | Carriage-Control Directives | 6-39 |
| Figure 6-6. | Carriage-Control Summary | 6-40 |
| Figure 7-1. | -LISTUSL Command Output | 7-13 |
| Figure 7-2. | -LISTRL Command Output | 7-21 |
| Figure 7-3. | -LISTSL Command Output | 7-26 |
| Figure 8-1. | Stack Dump Modes and selec Array Format | 8-61 |
| Figure 8-2. | Example of Abort Stack Analysis Dump | 8-68 |
| Figure 8-3. | ASCII vs 2741 Character Representation | 8-79 |
| Figure 8-4. | Echo Facility vs Duplex Mode | 8-82 |
| Figure 10-1. | Command Interpreter Error Messages | 10-3 |
| Figure 10-2. | Command Interpreter Warning Messages | 10-15 |
| Figure 10-3. | Program Errors | 10-20 |
| Figure 10-4. | Intrinsic Numbers vs Intrinsics | 10-21 |
| Figure 10-5. | Run-Time Errors | 10-22 |
| Figure 10-6. | File System Errors | 10-22 |
| Figure 10-7. | Loader Errors | 10-24 |
| Figure 10-8. | CREATE Errors | 10-25 |
| Figure 10-9. | ACTIVATE Errors | 10-26 |
| Figure 10-10. | SUSPEND Errors | 10-26 |
| Figure 10-11. | MYCOMMAND Errors | 10-26 |
| Figure 10-12. | LOCKGLORIN Errors | 10-26 |
| Figure 10-13. | System Messages | 10-28 |
| Figure 10-14. | Segmenter Error Messages | 10-32 |
| Figure 11-1. | Process Life Cycle | 11-2 |
| Figure 11-2. | Process Components and Tables | 11-4 |
| Figure 11-3. | Process Linking | 11-6 |
| Figure 11-4. | Process Deletion | 11-15 |
| Figure 14-1. | MPE/3000 Master Queue Structure | 14-7 |



PART 1
System Overview



SECTION I

Introduction to MPE/3000

The Multiprogramming Executive Operating System (MPE/3000) is a general-purpose, disc-based software system that supervises the processing of user programs submitted to the HP 3000 Computer. MPE/3000 relieves the user from many program control, input/output, and other housekeeping responsibilities by monitoring and controlling the input, compilation, run preparation, loading, execution and output of user programs. MPE/3000 also controls the order in which programs are executed, and allocates the hardware and software resources they require.

FEATURES

MPE/3000 offers the user many important features; some of these are found elsewhere only in medium to large-scale computers.

Multiprogramming

Through multiprogramming (interleaved processing), MPE/3000 allows numerous users to execute many different programs concurrently. The number of programs that can be processed concurrently depends on such factors as the hardware configuration, program operating modes (batch or interactive), and applications involved. Each programmer, however, uses the computer as if it were his own private machine — in other words, he need not depend on, nor even be aware of, others using the machine.

General-Purpose Versatility

MPE/3000 allows users to run batch and interactive programs concurrently.

BATCH PROCESSING. Batch processing lets programmers submit to the computer, as a single unit, commands that request various MPE/3000 operations such as program compilation and execution, file manipulation, or utility functions. Such a unit is called a *job*. Jobs contain all instructions to MPE/3000 and references to programs and data required for their execution; once a job is running, no further information is needed from the programmer. (Frequently, new programs and data are submitted as part of the job.)

Jobs are input through batch input devices such as card readers. In fact, several jobs can be submitted from multiple batch input devices concurrently. MPE/3000 schedules each job according to its priority. When a job enters execution, the commands within it are sequentially executed on a multiprogramming basis. MPE/3000 generates the job output on a local device such as a line printer, tape unit, or disc unit, or on a local or remote terminal. When one job is temporarily suspended, perhaps to await input of data, another (if available) immediately enters execution. Thus, when many jobs are active in the system, continuous processing and high throughput can be maintained.

INTERACTIVE PROCESSING. In interactive processing, the programmer interacts conversationally with the computer, receiving immediate responses to his input. Many users at different remote or local terminals can program on-line in this fashion. This type of interaction, called a *session*, can be used for program development, information retrieval, computer-assisted education, and many more applications where the user at a remote terminal must access the system directly.

MPE/3000 can continue to execute batch jobs at the same time it handles sessions. The basic difference between a session and a batch job is that a session is interactive, but a job is not. Thus, during a session, the user maintains a dialogue with the system to control input and monitor output; in a batch job, however, the command stream is entered into the system without a user/system dialogue.

Choice of Programming Languages

Under MPE/3000, the user can submit programs written in the following languages:

- FORTRAN/3000
- BASIC/3000
- COBOL/3000
- SPL/3000 (Systems Programming Language)

In addition, MPE/3000 supports subsystems for editing program files, performing statistical operations, aiding in debugging programs, running on-line diagnostic programs, and various other applications.

Each language translator and subsystem is accessed by a unique MPE/3000 command. The programmer need learn only one set of conventions for using these programs, because they all use the same general command formats, special characters, and error-diagnostic methods. Command coding is based upon the American Standard Code for Information Interchange (ASCII) Character Set, shown in Appendix A.

Operating Simplicity

MPE/3000 is easy to initialize, operate, monitor, and shut down. Its operation is overseen by a user with the MPE/3000 system manager capability, who assigns programming capabilities of various levels to each user. *Standard capabilities* allow the typical "general applications"

programmer to interact with the computer through a batch input device or a terminal. If this programmer is planning only to compile and execute a batch job, or to run an on-line interactive program, he may need to know only a few simple MPE/3000 commands. As his needs become more extensive, however, so must his knowledge of MPE/3000. A user planning more complex operations can be assigned various *optional capabilities*. These allow him to access more sophisticated system resources for such tasks as executing privileged instructions. Sets of capabilities are also used to protect the system and its users by limiting access to special system capabilities only to those who understand their correct use. Capability sets greatly simplify use of the system from the stand-point of each individual user — they define the extent to which he must understand and interrelate with MPE/3000, and permit a user to ignore aspects of MPE/3000 that do not apply to him.

Input/Output Conveniences

Because MPE/3000 treats all input/output devices as files (or groups of files in the case of mass storage devices), the programmer may access these devices by file names rather than by device types or logical unit numbers. The file names used in programs are independent of the devices used for file input and output, and need only be associated with these devices at the time the programs are run. This means that the user can write programs without immediate concern for the physical source of input or destination of output. This independence also means that programs can be run in either batch or interactive mode without changing the names of the files they reference.

Files on disc can be structured for either direct or sequential access, on an exclusive or shared basis. Direct-access files contain fixed-length records; sequential files, however, can contain either fixed- or variable-length records. For files on disc, storage space is automatically allocated as it is needed. MPE/3000 permits simultaneous access of sharable disc files by many programmers.

Processing Efficiency

MPE/3000 offers the user increased throughput by taking best advantage of the HP 3000 Computer architecture and the rapid speed of the central processor. This, in turn, permits the rapid changes between user environments that make multiprogramming in batch and interactive modes possible.

RE-ENTRANT CODE AND PRIVATE DATA. Within MPE/3000, many user and system functions can be active simultaneously without mutual interference. This is because the hardware provides protection of programs and guarantees the privacy of user data areas. MPE/3000 keeps code and data logically separate by organizing them into re-entrant code segments (which can be shared among users but not altered) and data segments (which cannot be shared but which can be altered by the creating user). *Code-sharing* means that only one copy of each program, accessible to many users concurrently, need exist in memory at any time. *Code segmentation* allows code to be moved from disc into main memory only when needed and dynamically relocated in main memory to accommodate other programs and routines. *Code re-entrancy* means that when a program is interrupted during execution of a code segment, and another program uses that same segment, the segment is completely protected from modification and will be returned, intact, to the previous program.

STACK ARCHITECTURE. Many powerful operating system features are made possible by the computer's use of stacks (linear storage areas for data) where the last item stored in is always the first item taken out. Some of these features are

- Ease of compilation and parameter passing
- Fast execution
- Highly-efficient subroutine linkage
- Minimum overhead
- Dynamic allocation of temporary storage
- Rapid interruption and restoration of user environments
- Code compression
- Recursion (where a procedure calls itself)

MICROPROGRAMMING AND THE CENTRAL PROCESSOR. Additional economy has been provided by microprogramming many system operations normally provided by software. These operations are requested by machine instructions that each, in turn, execute many micro-instructions built into the central processor hardware. Microprogramming eliminates the repetitive coding and main memory requirements otherwise needed for recurring operations (such as moving character strings from one location to another, or scanning strings for a particular character), thereby placing the operating burden on the highly-efficient micro-processor rather than on MPE/3000 software.

VIRTUAL MEMORY. The MPE/3000 virtual memory offers users a total memory space that far exceeds the maximum main-memory size of 131,072 bytes. Virtual memory consists of both main memory and an extensive, flexible storage area on disc. User programs and information in the disc area are subdivided into units (segments) of code or data that are dynamically moved, through overlays, into main-memory (core) for execution.

HIGH DATA THROUGHPUT. High throughput of data is facilitated by high-speed channels, double-buffering, many input/output devices, and input/output program execution (through an input/output processor) in parallel with regular program execution (by the central processor).

AUTOMATIC SCHEDULING. MPE/3000 automatically schedules all jobs and sessions according to their priorities. When execution of a running program is interrupted for any reason (such as input/output or an internal interrupt), MPE/3000 passes control to the program of highest priority awaiting execution.

SPOOLING. Spooling is a method whereby device contention can be reduced and better utilization of a device can be realized by buffering the data of the device on disc; the data is then processed/output at a later time. This facility is controlled by the console operator on a device basis, and generally has little functional impact on the user.

Accounting Facility

MPE/3000 keeps track of various system resources used by each job/session, group, and account; these resources include permanent file space, central-processor time, and (for sessions) terminal connect time. Limits can be set for the maximum use of these resources at the group or account level. As each job/session is logged off, the resource-use counters are updated. When another job/session attempts to log-on, and the central processor or (for sessions) terminal connect time limits have previously been exceeded, access is refused. When a request is made to save a file, and this action would result in exceeding the permanent file space limit at either the account or group level, the request is denied.

The accounting information for each group and account can be extracted and displayed, showing counts and limits for permanent file space (in disc sectors), central processor time (in seconds), and connect-time (in minutes).

Logging Facility

MPE/3000 enables a user with System Supervisor Capability to control the recording (on disc) of information about overall system activity. This log can be displayed through user-created analysis routines. The user can specify the information to be recorded on the log file, making his selection from the following areas: job/session and program initiation and termination, file closing, and system start-up and shut-down.

System-to-System Compatibility

All HP 3000 Computers operate under a single operating system — MPE/3000. This means that programs prepared on one HP 3000 can be run on any other without modification (provided that all devices required by those programs are connected on-line). It also means that users moving from one installation to another need not undergo additional training or read additional documentation to prepare themselves for the new environment.

Program and File Security

Each user operates in an environment protected from interference by other users. Program protection is provided by the hardware; file security is provided by a software facility based upon a series of lockwords and hierarchical access restrictions that allow the programmer to specify the degree of security desired.

System Manager and System Supervisor Capabilities

MPE/3000 provides many tools for overall management and control of the system to persons with the *System Manager* and *System Supervisor Capabilities*. These features are discussed in detail in *MPE/3000 Operating System, System Manager/System Supervisor Manual*.

The System Manager Capability allows a user to have final control of overall use of the system by defining the accounts under which users access MPE/3000, and the resource-use limits (if any) that apply to these accounts.

The System Supervisor Capability allows a user to supervise and control the general operation of the system by:

- Creating tapes for backing-up and modifying the system.
- Displaying certain system information.
- Permanently allocating/deallocating programs in virtual memory.
- Exercising greater scheduling control over processes than that allowed other users.
- Managing the system log file.

It is important to bear in mind that the above capabilities are granted to users through the assignment of special *attributes*, and in no way imply formal responsibilities or duties. Thus, a regular programmer may have the System Manager Capability or the System Supervisor Capability, or both capabilities — in addition to several others discussed later. For this reason, it is best to speak of “a user with System Manager Capability” rather than a formal “System Manager.”

SOFTWARE

MPE/3000 is furnished to the customer on a reel of magnetic tape. As part of the operating system software, these major components are provided:

- *System Configurator*, for configuring and making a back-up copy of MPE/3000.
- *System Initiator*, for installing and starting MPE/3000.
- *Command Interpreter*, for handling the commands that allow the user to interact with MPE/3000 through a terminal or batch input device.
- *File Management System*, for providing uniform access to disc files and standard input/output devices, and maintaining file security.
- *Memory Management System*, for dynamically allocating main-memory among contending users.
- *Dispatcher*, for allocating central-processor time among programs in execution.
- *Input/Output System and Drivers*, for scheduling, initializing, monitoring, and completing input/output requests for standard devices (accessed through the File Management System).
- *System Library*, for storage of frequently-used routines sharable among many users.

- *Segmenter Subsystem*, for segmenting and loading programs, and resolving references to external code.
- *Accounting System*, for maintaining and displaying resource usage counts.
- *Logging System*, for maintaining the system log file.

No additional or auxiliary software is required to install, operate, or maintain MPE/3000.

HARDWARE

The minimum hardware configuration required by MPE/3000 is

- Mainframe and Accessory Equipment Supplied, including HP 3000 Central Processor and 65,536 Bytes of Main Memory (Core), SIO Multiplexer Channel, System Control Desk (with System Console), and Asynchronous Terminal Controller
- Disc
- Magnetic Tape Unit

The disc is used to contain portions of MPE/3000 and user programs and data. The magnetic tape unit is used for cold-loading MPE/3000, loading subsystems (such as compilers), and storing user programs and data. (MPE/3000 is initialized and maintained through the console, which can also be used for input/output of user programs.)

The following optional hardware can be added to the system:

- Main Memory (for a total of up to 131,072 bytes)
- Extended instruction set
- Discs (Fixed-Head or Moving-Head)
- Magnetic Tape Units
- Card Readers
- Line Printers
- Card Punches
- On-Line Terminals
- Paper Tape Readers
- Paper Tape Punches
- Selector Channels
- Printing Reader/Punches
- Plotters
- Programmable Controller
- Synchronous Single-Line Controller

With this optional equipment, many hardware configurations are possible. For example, a small system used for batch processing might appear as shown in Figure 1-1. In this system, the card reader is used for input and the line printer for output, while the disc is used for storing system and user programs and data. The magnetic tape unit is used for cold-loading MPE/3000 and for storing user programs and data. The console is used to initialize and maintain MPE/3000.

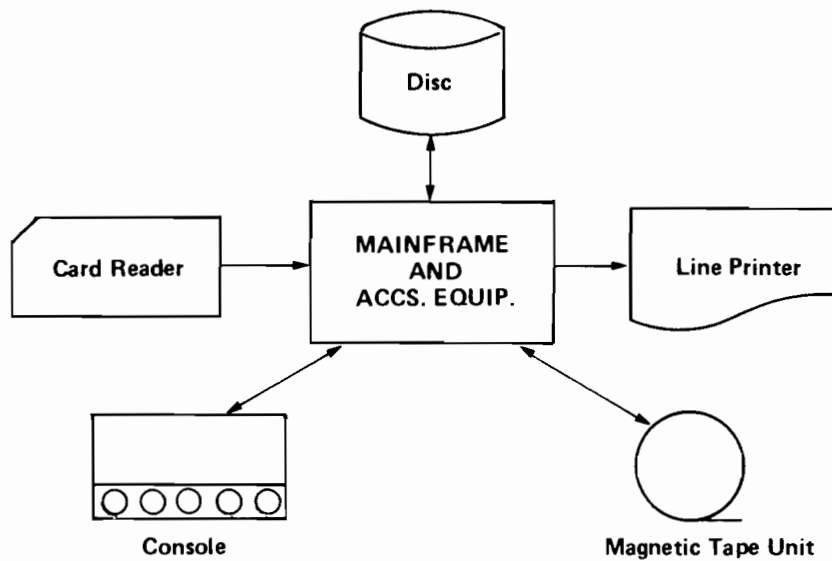


Figure 1-1. Small Batch System

A small interactive system might be configured as shown in Figure 1-2. In this system, up to seven terminals are used for input and output.

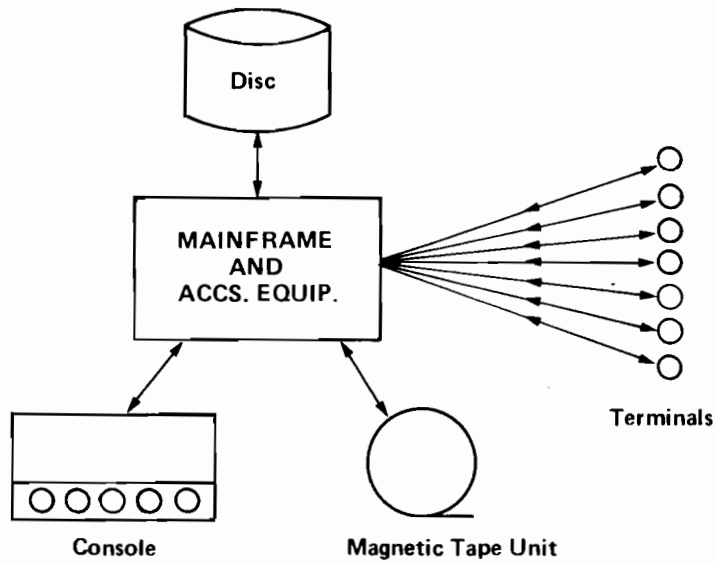


Figure 1-2. Small Interactive System

A combined batch and interactive system is illustrated in Figure 1-3. Here, batch and terminal operations can occur separately or concurrently.

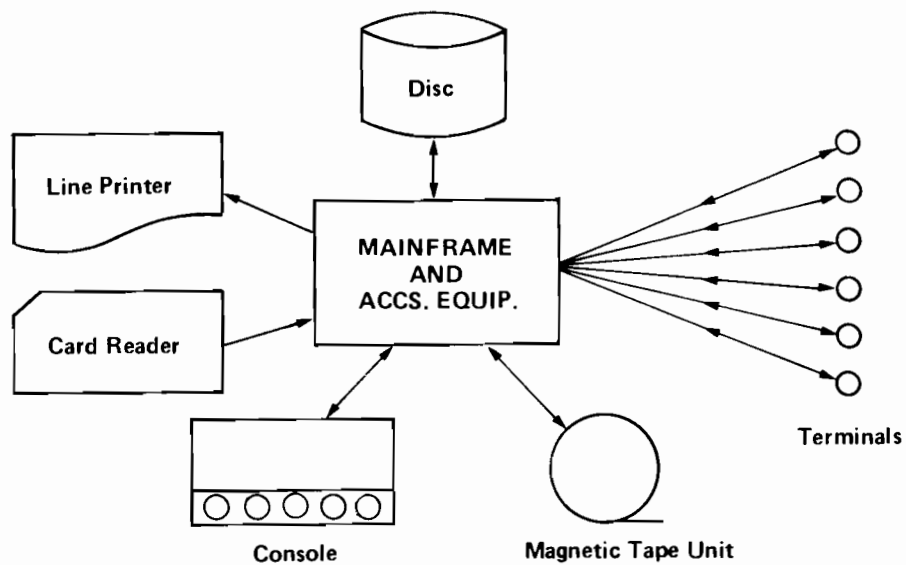


Figure 1-3. Combined Batch and Interactive System

A large processing system is shown in Figure 1-4. This system incorporates a large central memory, fixed-head and moving-head discs, many input/output devices for multiple-job batching, and eight terminals for remote processing.

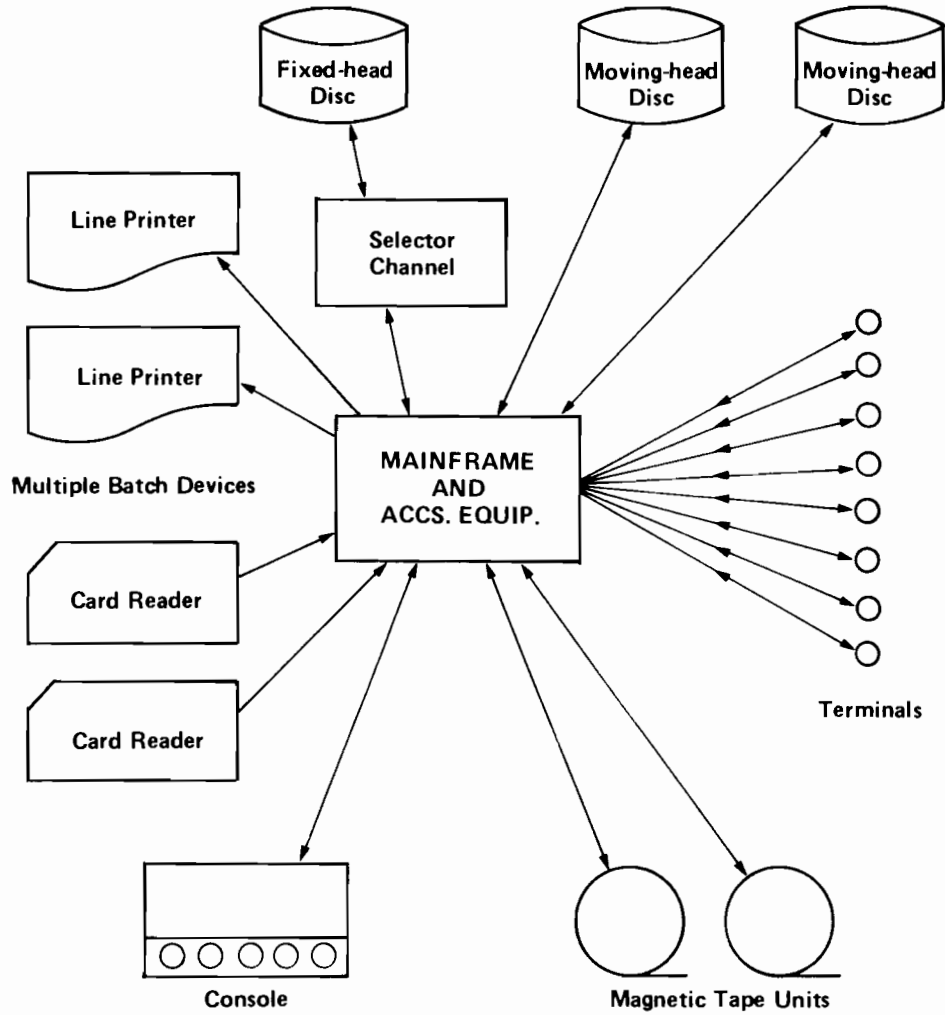


Figure 1-4. Large Processing System

SECTION III

Communicating with MPE/3000

To communicate with MPE/3000, the user issues commands and intrinsic calls.

Commands are requests issued to MPE/3000 to perform various broad functions external to the user's program. For example, they are used to initiate and terminate jobs and sessions, create and maintain files, compile and execute programs, call various utility subsystems, and obtain job status information. Commands can be entered through any standard input device, typically the card reader (for jobs) or the terminal (for sessions). Each command is accepted by the MPE/3000 Command Interpreter, which passes it to the appropriate procedure for execution. Following this execution, control returns to the Command Interpreter.

Intrinsic calls are used to invoke MPE/3000 functions requested within a user's program, such as reading, writing on, and updating files, skipping forward and backward on files, or returning system table information to the user's program. In an SPL/3000 program, the user writes the intrinsic calls explicitly. In FORTRAN/3000, BASIC/3000, or COBOL/3000 programs, for most applications, the compiler generates any necessary intrinsic calls automatically—they are invisible to the user. At their option, however, FORTRAN/3000, COBOL/3000, and BASIC/3000 users can also directly call intrinsics as their needs require, providing added power and flexibility to these standard programming languages.

The programmer can use intrinsic calls to invoke the Command Interpreter from within his program, and pass to it command images that will be interpreted and executed as the corresponding system commands.

The commands and intrinsic calls discussed in this part of the manual (Part 2) allow the user to initiate batch jobs and interactive sessions, access the file system, compile and execute programs, and use other *standard capabilities* of MPE/3000. They do not, however, permit him to handle processes, directly access the computer hardware, or use other *optional capabilities*; the commands and intrinsic calls for these capabilities are explained in Part 3.

COMMANDS

Each command entered by the user, whether in a batch job or interactive session, consists of

- A colon (used as a command identifier)
- A command name
- A parameter list (in most cases)

The end of each command is delimited by the end of the record on which it appears—for example, a *carriage return* for terminal input or the end of the card on which it is punched for card input. But, if the last non-blank character of the record is a *continuation character*, as defined later in this section, the end-of-record does not terminate the command. (Users running programs in batch mode should bear in mind that all 80 columns on each card image are scanned by MPE/3000, and thus no characters are ignored.)

The *colon* identifies a statement as an MPE/3000 command. In a batch job, the user begins each MPE/3000 command with this colon in column 1 of the source card (or card image). In an interactive session, however, MPE/3000 prints the colon on the terminal whenever it is ready to accept a command; the user responds by entering the command after the colon. (Interactive subsystems of MPE/3000 also use unique prompt characters; in a session, the prompt character output tells the user that a subsystem is ready.)

The *command name* requests a specific operation, and appears immediately after the colon. Imbedded blanks are not permitted within the command name. The end of the command name is delimited by any non-alphabetic character, normally a blank.

The *parameter list* contains one or more parameters that denote operands for the command. It is required in some commands but optional or prohibited in others. Parameter lists can include positional parameters and/or keyword parameter groups. Within the parameter list, delimiters (commas, semicolons, equal signs, or other punctuation marks) are used to separate parameters or parameter groups from each other, as described below.

Normally, the parameter list itself is separated from the command name by one or more blanks. However, when the first optional parameter in a positional list (as defined below) is omitted, the command name can be followed immediately by any other delimiter. (At the user's option, he can include one or more blanks between the command name and this delimiter.) Any delimiter can be optionally surrounded by any number of blanks, permitting a free and flexible command format.

EXAMPLE:

The following command (*RUN*) is preceded by a colon and includes a parameter list with two parameters:

```
:RUN PROG, ENTRYX
```

Both decimal and octal numbers are permitted as command parameters. Octal numbers, however, are always preceded by a percent sign (%).

Positional Parameters

With positional parameters, the meaning of the parameter is designated by its position in the list. For example, in the MPE/3000 command to compile a FORTRAN/3000 program, the parameter specifying an input file always precedes the one that specifies an output file. Positional parameters are mutually-separated by commas or semi-colons. The omission of an optional positional parameter from within a list is indicated by two adjacent delimiters; the

omission of a positional parameter that would otherwise immediately follow a command name is indicated by a comma or semi-colon as the first character in the parameter list. When parameters are omitted from the end of a list, however, no adjacent delimiters need be included to signify this — the terminating *return* or end-of-card is sufficient.

EXAMPLE:

The first command below has its first parameter omitted; the second command has its second (embedded) parameter omitted; the third command has its last two (trailing) parameters omitted; the fourth command has all parameters omitted. (In the second and third commands, the asterisk () is not a delimiter, but a special character used to denote a back-reference to a previously-defined file, as described in Section V. In each case, the asterisk is considered part of the following parameter.)*

```
:FORTRAN ,USFL,LISTFL,MFL,NFL
:FORTRAN *SOURCEFL,,LISTFL,MFL,NFL
:FORTRAN SOURCEFL,USFL,*LISTFL
:FORTRAN
```

A further example illustrates the relationship between the positions of parameters in a list and their meanings:

EXAMPLE:

In the following command (:FORTRAN), three positional parameters appear: INP refers to an input source file, OUT indicates an object (USL) output file, and LST indicates the listing output file. For the :FORTRAN command, these three fields always have the same meanings. (Note that 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.)

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

Keyword Parameters

When a parameter list is so long that use of positional parameters becomes difficult, keyword parameter groups are often used. The meaning of such a group is independent of its position in the list. A keyword parameter group is designated by a keyword that denotes its meaning, and *optionally* an equal sign and one or more sub-parameters. (Each keyword group is preceded by a semi-colon. When more than one sub-parameter appears in a group, they are usually separated from each other by commas. Like other delimiters, semi-colons and commas can be optionally preceded or followed by blanks.) With respect to each other, keyword groups can appear in any order. When keyword groups and positional parameters both appear in a list, however, the positional parameters always precede the keyword groups; when this occurs, and

trailing parameters are omitted from the positional group, their omission need *not* be noted by adjacent delimiters; instead, the occurrence of the first keyword indicates this omission.

EXAMPLE:

In this example, DL and CAP designate keyword parameter groups. PH, DS, and MR are sub-parameters of the keyword CAP.

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

Continuation Characters

When the length of a command exceeds one record (source card or entry-line), the user enters an ampersand (&) as the last non-blank character of this record and continues the command on the next record. In this case, the next record must begin with a colon (entered by the user in batch processing, but prompted by the terminal in interactive processing). Optionally, blanks can be embedded between the colon that begins a continuation record, and the rest of the information on that record. Commands can be continued up to 255 characters, including prompting colons and continuation ampersands.

EXAMPLE:

The following command image contains a continuation character at the end of the first line:

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

In continuing a command onto another line, the user cannot divide a primary command element (a command name, keyword, positional parameter, or keyword sub-parameter) — no primary element is allowed to span more than one line.

MPE/3000 does not begin interpretation of a command until the last record of the command is read. For interpretation, all records within the command are concatenated, and all prompt characters and continuation ampersands are replaced by one or two blanks.

Command Description Format

To help clarify the command descriptions that appear throughout this manual, system output is *underlined*. Input information is not underlined. (In cases where differences in output occur between batch and interactive processing, *interactive* output is assumed unless otherwise noted.) For both input and output, literal information that always appears exactly as shown is designated by *CAPITAL LETTERS AND SPECIAL CHARACTERS IN ITALICS*; on the other hand, symbols that represent variable information are indicated by *lower-case italics*.

Optional information is indicated by surrounding *[brackets]*. (However, the user does not enter these brackets as part of the command.) *{Braces}* indicate that one of the items included is required and must be entered by the user.

EXAMPLE:

The colon is output by the terminal during interactive sessions (as shown by the underline); BASIC is a literal command name entered by the user, (as indicated by capital letters); and commandfile, inputfile, and listfile are variable parameters input by the user (as shown by lower-case letters). All three parameters are optional (as indicated by brackets).

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

When more than one item is enclosed vertically in brackets, this indicates that exactly one of these items *may* be specified.

EXAMPLE:

```
_:SHOWJOB [ #  
              jsnumber  
              jsname ]
```

When more than one item is enclosed vertically in braces, this means one item *must* be specified.

EXAMPLE:

```
_:SAVE { $OLDPASS,newfilereference  
          tempfilereference }
```



If items are shown within vertical and disjoint brackets, this signifies that they are all optional and that those specified by the user can appear in any order.

EXAMPLE:

```
_:RUN progfile [,entrypoint]  
          [;NOPRIV]  
          [;LMAP]  
          [;MAXDATA = segsize]  
          .  
          .  
          .
```


Command Errors

All MPE/3000 commands issued, and any system messages for the user, are copied to the job/session listing device.

When MPE/3000 detects an error in a command, it suppresses execution of that command and prints an error message. If this occurs during a batch job, (and no :CONTINUE command precedes the erroneous command, as discussed later) all information between the erroneous command and the end of the job is ignored, and the job is aborted. If an error occurs during an interactive session, an indication is printed and control returns to the user; he may then re-enter this command correctly, or enter any other command he desires simply by pressing the carriage-return key and entering the command. Alternatively, if he desires a fuller explanation of the error, he can request it by entering any character directly after the error-number shown in the error message. In response, the explanation appears on the next line. Error messages and on-line error recovery are discussed in Section X.

INTRINSIC CALLS

A major advantage of MPE/3000 intrinsics is that they can be called directly from programs written in standard, higher-level programming languages (FORTRAN/3000, COBOL/3000, and BASIC/3000) as well as from SPL/3000 programs. The rules for invoking intrinsics from programs written in standard languages are presented in the manuals covering those languages. However, because intrinsic calls are issued primarily by users programming in SPL/3000, the rules for issuing these calls from SPL/3000 programs are summarized below.

Before an intrinsic can be called from an SPL/3000 program, it must be declared within the declaration portion of the program, following all data declarations, like any other SPL/3000 procedure. This can be done by writing the standard PROCEDURE declaration. Normally, such a declaration would contain both the procedure head and the procedure body (code). But, since an intrinsic is an *external* procedure, the user follows the normal SPL/3000 conventions for such procedures by writing only the intrinsic *head* and including within it the OPTION EXTERNAL notation. (EXTERNAL signifies that the body code will be supplied from a library and linked to the user's program during program allocation.) In this manual, the complete head format for each intrinsic is presented with the discussion of that intrinsic. (Note that in SPL/3000, each statement is delimited by semi-colons.)

EXAMPLE:

The format for the head of the FREAD (file read) intrinsic is presented in Section VI. Using this format, the user could write a declaration that will enable him to call the FREAD intrinsic later in his program. The declaration could appear as follows. Note that the user must include the OPTION EXTERNAL notation.

```
INTEGER PROCEDURE FREAD (FN,TAR,TC);  
VALUE FN,TC;  
INTEGER FN,TC;  
ARRAY TAR;  
OPTION EXTERNAL;
```

Because some intrinsic heads are rather long, the SPL/3000 programmer can save time and effort by using an alternative method of declaring intrinsics; the INTRINSIC declaration. This is written in the following format

INTRINSIC intrinsicname, intrinsicname, . . . , intrinsicname;

In the *intrinsicname* list, the user names all intrinsics he intends to call within his program. (When more than one is named, the names are separated by commas.) He then need not write the heads for these intrinsics.

EXAMPLE:

To use the INTRINSIC declaration to declare the FOPEN, FREAD, FWRITE, and FCLOSE intrinsics, the user could write:

INTRINSIC FOPEN,FREAD,FWRITE,FCLOSE;

Regardless of whether the programmer declares an intrinsic with a PROCEDURE declaration or an INTRINSIC declaration, he must know the formal head format for each intrinsic, since this tells him the number and type of parameters that are used in calling that intrinsic.

The user calls an intrinsic in his program exactly as he calls any SPL/3000 procedure: he writes the intrinsic name and a parameter list, enclosed in parentheses. These elements follow the positional format shown in the intrinsic head; parameters are separated from each other by commas.

EXAMPLE:

A call to the FREAD intrinsic could be written as

FREAD (INFILE,BUFFER,-80);

If the OPTION VARIABLE notation appears in the intrinsic head, some of the intrinsic parameters are optional. In this manual, these optional parameters are indicated as bold face in the intrinsic head formats.

Since all intrinsic parameters are positional, the user indicates a missing parameter within a list by following the previous delimiting comma with another comma by itself.

EXAMPLE:

The second parameter is missing:

FOPEN (FILENAME,,3);

If the first parameter is omitted from a list, this is indicated by following the left parenthesis with a comma. If one or more parameters are omitted from the end of a list, this is indicated by simply writing the terminating right parenthesis after the last parameter included.

NOTE: *In some intrinsic calls, input parameters are passed to the intrinsic as words whose individual bits or fields of bits signify certain functions or options. In cases where some of the bits within a word are described in this manual as "not used by MPE/3000," the user is advised to set such bits to zero. This will help ensure the compatibility of his current programs with future releases of MPE/3000.*

In cases where output parameters are passed by an intrinsic to words referenced by a users' program, bits within such words that are described as "not used by MPE/3000" are set as noted in the discussion of the particular parameter.

All callable intrinsics except **DEBUG** reset the hardware Carry bit, in addition to modifying the condition code. (**DEBUG** is discussed in Section VIII.)

Intrinsic Description Format

In this manual, the complete intrinsic declaration head is shown with the discussion of each intrinsic; included within this is the intrinsic call format, distinguished from the remainder of the head by a box. This format appears as follows:

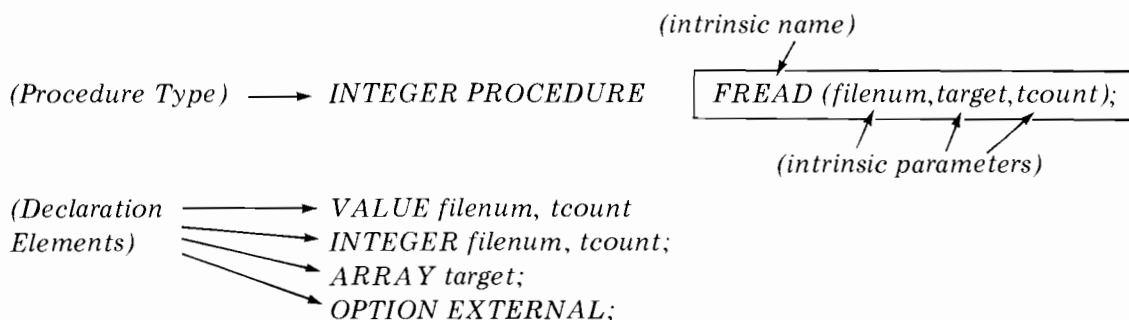
procedure type **PROCEDURE** *intrinsic name (parameter list)*
declaration elements

The *procedure type* applies only to *type intrinsics*—those that return a value to the user's program in response to the intrinsic call. This describes the type of data returned, such as integer, real, or byte values. The data is returned through the intrinsic name, as demonstrated in later examples. Additionally, type intrinsics can be directly related to functions called in FORTRAN/3000 programs, as noted in the *FORTRAN/3000* manual.

The *declaration elements* describe each parameter (variable, pointer, label, and array) in the parameter list, and its characteristics. If the **VALUE** notation is present in the declaration, either a numeric value (expression) or a symbolic identifier may be specified for the parameters indicated by this notation. If **VALUE** is not present, only symbolic identifiers may be used for parameters.

EXAMPLE:

This is the intrinsic description format for the **FREAD** intrinsic:



Intrinsic Call Errors

Some intrinsics alter the *condition code*, made available to the user's FORTRAN/3000 and SPL/3000 programs through the Status register. Since the contents of this register change continually, the user should check this register for condition codes immediately upon return from an intrinsic. A condition code is always one of the following, and has the general meaning indicated. The specific meaning, of course, depends upon the intrinsic called.

| Condition Code | General Meaning |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CCE | Condition code is zero. This generally indicates that the user's request was granted. |
| CCG | Condition code is greater than zero. A special condition occurred but may not have affected the execution of the user's request. (For example, the request was executed, but default values were assumed as intrinsic call parameters.) |
| CCL | Condition code is less than zero. The user's request was <i>not</i> granted, but the error condition may be recoverable. Beyond this condition code, some intrinsics return further error information to the user's program through their return values. |

Two types of errors may be encountered when an intrinsic is executed. The first, denoted by the CCG or CCL condition codes, is generally recoverable and is known as a *condition-code error*. The second type is an *abort error*, which occurs when a calling program passes illegal parameters to an intrinsic, or does not have the capability class demanded by the intrinsic. The user's program may recover from this error or take some other appropriate action if an appropriate error-trap procedure has been defined (as discussed in Section VIII). Otherwise, his program terminates, and an abort-error message appears on his output device. If the program was entered in a batch job, MPE/3000 removes the job from the system (unless a :CONTINUE command, defined later in this section, precedes the error); if it was entered in an interactive session, MPE/3000 returns control to the user at the terminal. Abort-error messages are described in Section X.

NOTE: Whenever a callable intrinsic is invoked by a process running in either the non-privileged mode, or the privileged mode with the DB register pointing to the DB area in the user's stack, a bounds check takes place to ensure that all parameters in the intrinsic call reference addresses that lie between the DL and S addresses in the stack (prior to the intrinsic call); if an address outside of these boundaries is referenced, an abort-error occurs.

When a callable intrinsic is invoked by a process running in the privileged mode, and the DB register points to a data segment other than the user's stack segment, the results depend on the particular intrinsic. Most intrinsics immediately abort in this case. Others (indicated in Appendix C) are allowed to execute following a

bounds-check that ensures that all parameters in the intrinsic call reference addresses that lie within the data segment; any boundary-violation results in an abort-error. Any additional special actions taken by a particular intrinsic are described in the discussion of that intrinsic.

BATCH JOBS

The user can initiate and terminate batch jobs through any device that accepts serial input (such as a card reader, tape unit, or terminal used non-interactively) located at the computer site, or through a terminal used non-interactively and located remotely, as discussed below.

Initiating Batch Jobs

A batch job consists of a set of cards (or card images) that begins with a :JOB command, contains additional MPE/3000 commands, user programs, and optional data, and terminates with an :EOJ command. The :JOB command initiates the job by establishing contact with MPE/3000. If it accepts this command, MPE/3000 begins job processing. When processing begins, commands are sequentially accepted from the job until an :EOJ command is encountered or until the job is halted by one of the abnormal events described under "Premature Job or Session Termination."

A job can be entered through any device configured to accept jobs. The user (or computer operator) first requests an interrupt on the job input device as follows:

1. For card readers or magnetic tape units, the user turns the device on-line.
2. For terminals, the user turns the terminal on, dials MPE/3000 (if the terminal is connected over switched telephone lines), and presses the *return* key (to generate a carriage return).

NOTE: The device through which a job/session is entered is called the job/session input device, and is recognized as the standard input file for the job/session. Each potential job/session input device is related, during system configuration, to a corresponding job/session listing device that is recognized as the standard output file for the job/session (unless another device is designated by the user for this purpose). There is one job/session input device and one corresponding job/session listing device for each job/session.

The user writes the :JOB command in the following format. (The keyword parameter groups, of course, can be specified in any order.)

```
:JOB  [jobname,] username[/upass].acctname[/apass] [,groupname[/gpass]]
      [;TERM=termtype]
      [;TIME=cpulimit]
      [;PRI=executionpriority]
      [;HIPRI
      [;INPRI=inputpriority]
      [;RESTART]
      [;OUTCLASS=[device] [, [outputpriority] [, numcopies]]]
```

The parameters have the meanings noted below.

NOTE: In MPE/3000, names (such as the *jobname*, *username*, *upass*, *acctname*, *apass*, *groupname*, and *gpass* parameters noted below) consist of from one to eight alphanumeric characters, beginning with a letter, unless otherwise specifically indicated. Embedded blanks are not permitted within *username.accountname* specifications.

jobname An arbitrary name used in conjunction with the *username* and *acctname* parameters to reference this job in other commands. If omitted, a *null jobname* is assigned. (Optional parameter.)

NOTE: A fully-qualified *jobname* consists of [jobname.]username.acctname.

username The user's name, as established in MPE/3000 by the user with Account Manager Capability. This name is unique within the account. (Required parameter.)

upass The user's password. (Required if the user has been assigned a password by the user with Account Manager Capability.)

acctname The name of the user's account, as established by the user with System Manager Capability. Note that this parameter is preceded by a *period* as a delimiter. (Required parameter.)

apass The account password. (Required if the account has been assigned a password by the user with System Manager Capability.)

groupname The name of the group (established by the user with Account Manager Capability) that the user wishes to use, during his job, for his local file domain and for central processor time accumulation charges. If omitted, MPE/3000 assigns the user's home group. (Optional parameter for users with a home group; Required parameters for users without a home group.)

gpass The group password. When a user logs on under his home group, no password is needed. (Required parameter for some groups.)

termtype The type of terminal used for input.

(MPE/3000 uses the *termtype* parameter to determine device-dependent characteristics such as delay factors for carriage returns.) This parameter is indicated by one of the numbers 0 through 8, with the meanings shown below:

0 = ASR 33 EIA-compatible HP 2749B (10-characters per second (cps)).

0 = ASR 35 EIA-compatible (10 cps).

1 = ASR 37 EIA-compatible (15 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, 30, 60 cps).

6 = HP 2762A or HP 2762B (GE Terminet 300 or 1200), or Data Communication Terminal Model B (DCT 500) (10, 15, 30, 120 cps).

7 = Selectric (IBM 2741) CALL 360 Correspondence Code (14.8 cps).

8 = Selectric (IBM 2741) (PTTC/EBCD Code) (14.8 cps).

9 = HP 2615A (MiniBee) (10-240 cps).

10 = HP 2640A Interactive Display Terminal (Character mode only) (10-240 cps).

NOTE: Users at IBM Selectric terminals are directed to the *NOTE* under the discussion "Initiating Sessions," later in this section.

(Additional information on terminals appears in Section VIII.)

If the *termtype* parameter is omitted, and a terminal is used, a default of 0 is assigned. (Optional parameter for ASR 33 or ASR 35 terminals; Required parameter for all others, to ensure correct listings.)

cpulimit

Maximum central-processor time permitted for the job, entered in seconds. If this limit is reached during the job, the job is aborted. If a question mark is entered, no limit applies. The maximum limit entry is 32767. If omitted, an installation-defined limit applies. (Optional Parameter.)

NOTE: The *cpulimit* parameter should not be confused with the central-processor time limit defined for groups and accounts. The *cpulimit* parameter (or its default value) applies to each individual job, and aborts the job as soon as it is exceeded. The central-processor time limit for groups and accounts, however, limits the total central-processor time used by all jobs that have run under the particular group and account--it is checked only at log-on time, and can only terminate access at that time; it will never cause a job in process to abort.

executionpriority

The priority (subqueue) desired for MPE/3000 command interpretation, and also the default priority for all programs within the job. For users with *standard capabilities*, this may be CS, DS, or ES, indicating the priorities (subqueues) described in Section II. For users with optional capabilities, this may be any of the priorities described in Part 3 (depending on the maximum subqueue priority permitted this user). If the priority specified exceeds that permitted for this user, the highest priority possible below BS applies. If this parameter is omitted, a default priority established by the System Supervisor is assigned. (Optional Parameter.)

HIPRI

A request for maximum job-selection priority. It can only be specified by System Managers and System Supervisors. This parameter will cause the job to be dispatched regardless of the current job fence or execution limit. (Optional Parameter.)

inputpriority

The relative priority to be used for selecting this job when several jobs of equal *executionpriority* are ready to begin processing. This parameter is one or two digits, ranging from 1 (lowest priority) to 13 (highest priority). The default value is 13 (unless logging is on; see System Manager/Supervisor Manual). If the supplied value is less than the current job fence set by the console operator, the job will be deferred until the operator lowers the job fence or raises the job's *inputpriority*. (Optional Parameter.)

RESTART

(Meaningful only for jobs initiated on spooled input devices.) Indicates that this job should be automatically restarted on a cold load following a system failure which occurred during the job's execution (normally such jobs would have to be re-submitted). This parameter is ignored for jobs on non-spooled input devices. (Optional Parameter.)

- device* A particular device (such as a specific line printer) to be used for listing output. This parameter is a device class name or a logical device number (as defined in Section V), and may only be entered by users who have the *non-sharable input/output device* file-access attribute. If this parameter is omitted, the device assigned is the one defined (during system configuration) as the default job/session listing device that always corresponds to the user's current input device. Magnetic tape units cannot be accepted as the output device for a job. (Optional Parameter.)
- outputpriority* (Meaningful only for jobs whose listings are directed to spooled output devices.) The "output priority" to be attached to the job list file destined for a spooled line printer or card punch. This priority is used by the system to select the next file to be output, among all those spooled files waiting for a particular device. It consists of a digit between 1 (lowest) and 13 (highest), inclusive. This parameter is ignored when the listing is directed to a non-spooled device. The default value is 13 (unless logging is on; see System Manager/Supervisor Manual). (Optional Parameter.)
- numcopies* (Meaningful only for jobs whose listings are directed to spooled output devices.) The number of copies of the job list file to be produced. This number must be less than 256. This parameter is ignored when the listing is directed to a non-spooled device. The default value is 1. (Optional Parameter.)

The *upass*, *apass*, and *gpass* parameters, when included, must be separated from their preceding parameters with a slash. (The slash should not be preceded nor followed by blanks.) When the job list device is not the job input device, passwords are not printed on the job listing.

EXAMPLE:

This job is named ALPH22, and is entered under the user name RJOHNSON, account name ACCT1003, and the group GPA2. The group password MG03 is used. A central processor time limit of 300 seconds is entered. An execution priority of DS is requested. For all other parameters, default values apply. Notice the continuation character (&) at the end of the first line.

```
:JOB ALPH22,RJOHNSON.ACCT1003,GPA2/MG03; TIME=300; &
:PRI=DS
```

If the central-processor time already accumulated by previous jobs exceeds the total time allotted for the user's account or group, the job is rejected upon submission. (Note that these account/group limits are checked only at the start of a job.) Otherwise, the job runs until completed or until aborted because of an error or because the limit designated by the *cpulimit* parameter in the :JOB command is reached.

All MPE/3000 commands encountered during job processing are listed on the job listing device (typically, a *printer*). Acceptance of the job is indicated by information output in the following format, immediately after the listing of the :JOB command.

JOB NUMBER = #Jnnnnn
date, time
HP32000v.uu.ff

nnnnn Job number assigned by MPE/3000 to uniquely identify the job to the system. This may range from one to five digits long. The user can reference the job in subsequent commands by this number or by the job identification specified through the :JOB command:

[jobname,] username.acctname

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

time Current time (hours:minutes am/pm)

v MPE/3000 version level.

uu MPE/3000 update level.

ff MPE/3000 fix level.

EXAMPLE:

If the job in the previous example had been assigned the job number 7, and had been submitted at 3:23 p.m. on January 31, 1972, the job output listing would appear as follows (with the beginning of the acceptance message shown here by an arrow):

```
➡ :JOB ALPH22,RJOHNSON.ACCT1003, GPA2/MG03; TIME=300; &  
:PRI=DS  
JOB NUMBER = #J7  
MON, JAN 31, 1972, 3:23 PM  
HP32000B.02.08
```

Terminating Batch Jobs

The :EOJ command terminates the batch job. This command has no parameters and is written simply as

:EOJ

When this command is encountered, MPE/3000 acknowledges job termination by printing the following information on the job listing

CPU(SEC) = cputime
ELAPSED (MIN) = elapsedtime
date, time
END OF JOB

| | |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------|
| <i>cputime</i> | Total central processor time used by job, in seconds, charged to account and group. |
| <i>elapsedtime</i> | Total wall-clock time between the beginning and end-of-job, in minutes. This value is <i>not</i> charged to the account and group. |
| <i>date</i> | Current date (day-of-week, month and day, year) |
| <i>time</i> | Current time (hours:minutes am/pm) |

If no :EOJ command is included to terminate the current job, the next :JOB command will have one of the following effects:

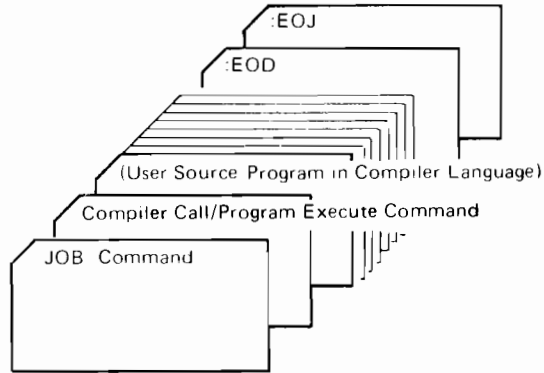
1. If read by the MPE/3000 Command Interpreter, it will terminate the current job and start a new one.
2. If read by a user's program, it will signal an end-of-file to the program but will be ignored by MPE/3000. (Because this can occur, the user should be sure that he terminates his job with an :EOJ command.)

Typical Job Structures

All batch jobs must begin with a :JOB command and terminate with an :EOJ command. Additionally, the end of each program input within a job should be indicated with an :EOD command, (written simply as :EOD). Beyond this, job structures can vary considerably from application to application.

EXAMPLE:

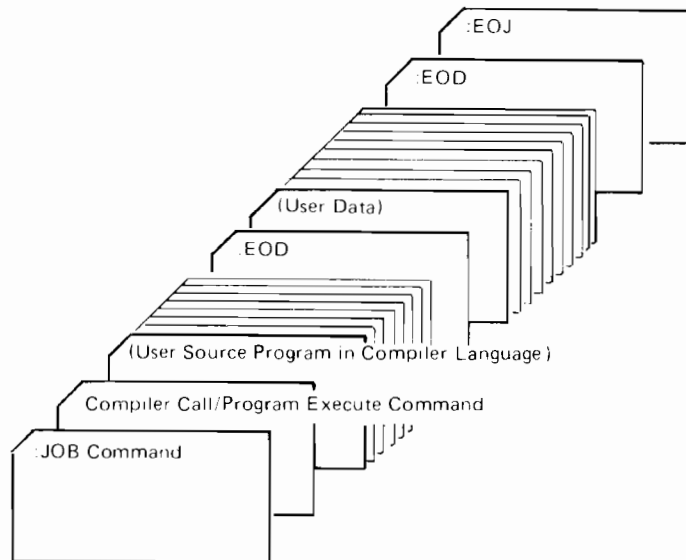
A job submitted through the card reader to compile and execute a program might appear as follows. (Note that an :EOD card denoting the end of the program must precede the :EOJ card.)



When data is included within a job, its end must also be indicated with an :EOD command. (Users writing language processors and other subsystems should be aware that MPE/3000, when reading input from the standard input stream, interprets *any* record beginning with a colon as the end-of-data, whether or not this record contains an :EOD command. The :EOD command is used as a data delimiter in general applications because it executes no *additional* functions.)

EXAMPLE:

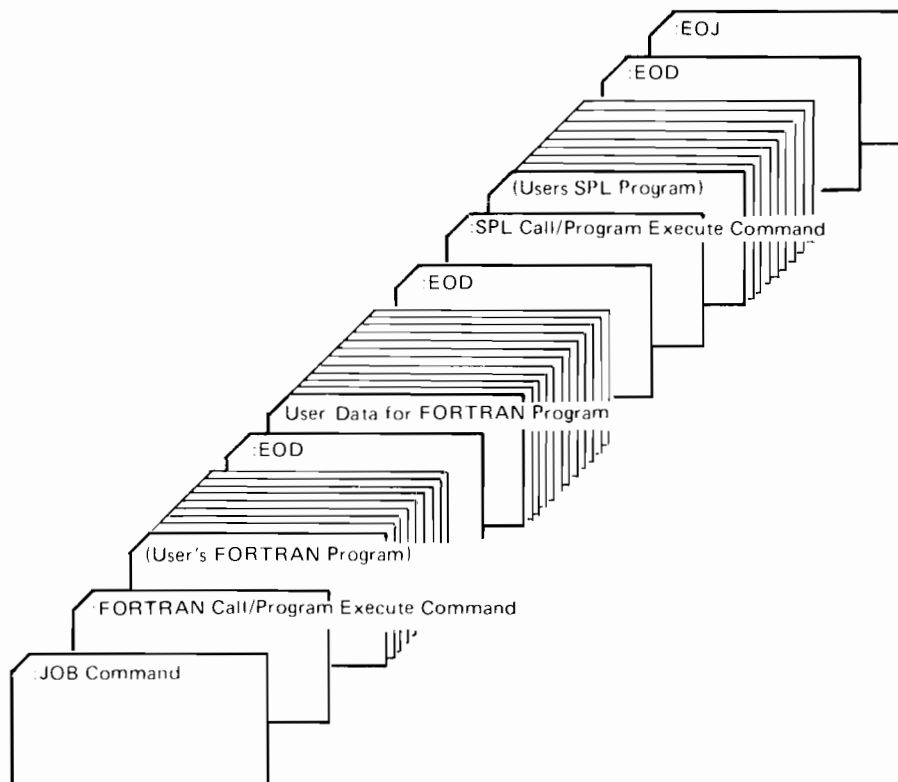
The job shown below includes data to be processed during program execution:



Of course, jobs with much more elaborate structures are possible. Such jobs may contain several user programs, input to different compilers, obtaining input data from various sources, and transmitting output data to several devices:

EXAMPLE:

The following job contains: a FORTRAN/3000 program to be compiled and executed; data for that program; and an SPL/3000 program to be compiled and executed, with data input from a disc file.



INTERACTIVE SESSIONS

The user initiates and terminates interactive sessions from a terminal. (Non-interactive devices cannot be used for this purpose.)

Initiating Sessions

The user initiates a session by turning the terminal on, dialing MPE/3000 (if the terminal is connected over switched telephone lines), pressing the *return* key to generate a carriage return and produce the first prompt character (colon), and entering the :HELLO command described below. This logging-on establishes contact with MPE/3000 and allows the user to enter commands to compile and run programs or request other MPE/3000 standard capabilities. From this point on, the user is automatically prompted for each command (and thus does not type the colon). The session exists until the user issues a command to terminate it, or until one of the conditions described under "Premature Job or Session Termination" occurs.

NOTE: Users at IBM 2741 Selectric Terminals (PTTC/EBCD Code or Call 360 Correspondence Code) must always type H or h as the first character of their input following the initial prompt character; this enables MPE/3000 to determine (from this character's bit pattern) the type of Selectric being used.

Users at IBM 2741 Selectric Terminals (CALL/360 or PTTC/EBCD code) should note that any CALL/360 or PTTC/EBCD character that does not have an equivalent ASCII character is ignored on input.

Users at IBM 2741 Selectric Terminals (Call 360 Correspondence Code) will receive a not-equals sign (≠) as the first prompt character.

The user enters the :HELLO command in the following format

```
_:HELLO [sessionname,] username[/upass] .acctname[/apass] [,groupname [/gpass] ]  
    [;TERM = termtype]  
    [;TIME = cpulimit]  
    [;PRI = executionpriority]  
    [;HIPRI  
    [;INPRI=inputpriority]
```

The parameters have the meanings noted below.

NOTE: The sessionname, username, upass, acctname, apass, groupname and gpass parameters are all names that can contain up to eight alphanumeric characters, beginning with a letter.

sessionname An arbitrary name used in conjunction with the *username* and *acctname* parameters to reference this session in other commands. If omitted, a null *sessionname* is assigned. (Optional parameter.)

| | | |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><i>username</i> <i>upass</i> <i>acctname</i> <i>apass</i> <i>groupname</i> <i>gpass</i> <i>termtype</i> <i>executionpriority</i> <i>HIPRI</i> <i>inputpriority</i></p> | } | <p>The same definitions noted under the corresponding parameters in the :JOB command, applying to <i>sessions</i> rather than jobs. The default <i>executionpriority</i> for sessions is CS.</p> |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

cpulimit The same definition noted for *cpulimit* in the :JOB command, applying to *sessions*, and with this exception: if the *cpulimit* parameter is omitted, *no* limit applies.

The *upass*, *apass*, and *gpass* parameters, when included, must always be separated from their preceding parameters by a slash. (This slash should not be bounded by blanks.)

EXAMPLE:

In the following command, the user establishes a session named *INTER3*, under the user name *JONES*, account name *ACT20*, and *GROUP 3*. The account password of *PASS* is also entered.

:HELLO INTER3,JONES.ACT20/PASS, GROUP3

When a user is beginning a session through a terminal connected over switched telephone lines (a dial-up terminal), he must specifically log on within a period defined during system configuration. Otherwise, his terminal is disconnected and he must dial MPE/3000 again.

When entering a session through certain full duplex terminals, the user can suppress the printing (echoing) of passwords at his terminal by temporarily requesting half-duplex mode (by pressing the *ESC* and ; keys). (The user returns to full-duplex mode by pressing the *ESC* and : keys.) Where the terminal does not permit dynamic half-duplex operation (such as IBM 2741 Selectric Terminals, or any terminal except HP 2600A connected over the 202 modem), the user can conceal the passwords simply by omitting them from the command; in this case, after the :HELLO command is echoed, MPE/3000 outputs a corresponding request for each required password, directs the carriage to skip one line, prints an eight-character mask, and returns the carriage to the beginning of the line, as shown:

| | | |
|----------------------------------------------------------------------------|---|-------------------------|
| <p>{ <u>USER</u> <u>ACCOUNT</u> <u>GROUP</u> >}</p> | } | <p><u>PASSWORD?</u></p> |
| <p><u>NNNNNNNN</u> ← (mask)</p> | | |
| <p>↑ (carriage return)</p> | | |

The user then enters the password. The password is echoed on the input/listing device, but is printed on top of the mask to ensure privacy.

EXAMPLE:

Suppose that a group password of *ORG* was required, but not entered, in the command shown in the previous example. The user would be prompted for the password, and would enter it (over the mask) as follows:

```
:HELLO INTER3,JONES.ACT20/PASS, GROUP3  
GROUP PASSWORD?  
▶ XXXXXXXXXX
```

For sessions in which input and listing output are generated on separate devices, MPE/3000 omits the password entered.

Upon initiation of the session, the following information is transmitted to the terminal:

```
SESSION NUMBER = #Snnnnn  
date, time  
HP32000v.uu.ff
```

nnnnn Session number assigned by MPE/3000 to uniquely identify the session. This may range from one to five digits long.

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

time Current time (hours:minutes am/pm)

v MPE/3000 version level.

uu MPE/3000 update level.

ff MPE/3000 fix level.

EXAMPLE:

Suppose that the session in the previous example is accepted by MPE/3000 under the session number 512 at 12:37 p.m. on May 5, 1972. The following information would be output. (Remember, the mask is printed before the password may be entered.)

```
:HELLO INTER3,JONES.ACT20/PASS, GROUP3
GROUP PASSWORD?
XXXXXXXXXX
▶ SESSION NUMBER = #5512
FRI, MAY 5, 1972, 12:37 PM
HP32000R.02.08
```

If the connect time or the central processor time limit allotted to the account or group specified by the :HELLO command has been exceeded by previous sessions or jobs, the current session is rejected at log-on. If these limits expire during the session, however, the session is allowed to continue until terminated by the user; the next session under this account or group, however, will be rejected at log-on.

If the user enters an illegitimate :HELLO command, the following error message appears:

:ERR 8

At this point, a request to re-enter the :HELLO command appears. (The erroneous element in the previous :HELLO command is not identified.)

NOTE: A system power or line failure automatically results in the permanent termination of sessions; the sessions must be reinitiated from the beginning.

Interrupting Program Execution Within Sessions

The user can interrupt execution of a program or subsystem at any point in a session by striking the BREAK key. (On some terminals this is labeled "INTERRUPT" or "ATTENTION".)

This returns control to the MPE/3000 Command Interpreter, allowing the user to issue commands to abort the program (without disrupting the session), perform various file or utility operations, or resume the program. Appendix B, under the heading "BREAKABLE COMMANDS", summarizes which commands are breakable and which commands may be issued while in break mode.

To abort the program, the user enters this command:

:ABORT

To request an operation (such as creating or deleting a file), the user enters the command for that operation. File and utility commands are executed immediately. Some commands, however, require aborting of the user's program before they can be executed. When the user enters such a command during a program break, MPE/3000 prints:

ABORT?

The user responds by entering *YES* (to abort the current program and execute the command) or *NO* (to return control to the Command Interpreter).

To resume a program at the point where it was interrupted, the user enters:

:RESUME

If a read operation was pending for the program on the terminal when the break occurred, the following message is output:

READ PENDING

The user must satisfy the pending read before program execution can resume. (All characters entered before the break are retained.)

NOTE: The :ABORT and :RESUME commands are legitimate only during a break.

Terminating Sessions

To terminate a session, the user enters the following command:

:BYE

MPE/3000 acknowledges termination by printing the following information:

CPU (SEC) = cputime
CONNECT (MIN) = connecttime
date, time
END OF SESSION

| | | |
|--------------------|---|-------------------------------------------------------------------------------------------------|
| <i>cputime</i> | = | Total central-processor time used by the session, in seconds, logged against group and account. |
| <i>connecttime</i> | | Total wall clock time between log-on and log-off, in minutes, logged against group and account. |
| <i>date</i> | | Current date (day-of-week, month and day, year) |
| <i>time</i> | | Current time (hours:minutes am/pm) |

EXAMPLE:

Typical terminal output resulting from logging off:

```
:BYE  
  
CPU (SEC)=4  
CONNECT (MIN)=2  
THU, MAY 7, 1973, 12:56 PM  
END OF SESSION
```

For accounting purposes, the connect time is rounded upward to the nearest minute, and the central processor time is shown in seconds. Both sums are added to the usage counters for the session's account and group.

If the user hangs up the receiver at a dial-up terminal, an implicit :BYE command is issued and the session terminates.

Typical Session Structure

All sessions must begin with a :HELLO command. They typically terminate with a :BYE command. The end of data within a session must be indicated with the :EOD command.

If a user issues a second :HELLO command within his current session, this results in an implicit :BYE and :HELLO sequence - that is, it terminates the current session and starts another one.

The structure of sessions varies with the application. One example follows.

EXAMPLE:

```
:HELLO (Session, user, and account identification.)  
SESSION NUMBER = #Snnn  
(date) (time)  
HP3200v.uu.ff  
  
:(MPE/3000 Command to call BASIC Interpreter.)  
.  
.  
.  
(User Program in BASIC.)  
(User Command to exit from BASIC Interpreter.)  
:BYE  
  
CPU (SEC) = cputime  
CONNECT(MIN) = connecttime  
date, time  
END OF SESSION
```

READING DATA FROM OUTSIDE STANDARD INPUT STREAM

Users can designate, for a job or session, input consisting only of data to be read from a non-sharable, auto-recognizing device that is *not* the standard input device for that job or session. (An auto-recognizing device is one that automatically accepts the :JOB, :HELLO, or :DATA commands.) This designation is typically made to read a deck from a card reader while in session mode. Indeed, it is required for every data deck not imbedded in the standard input stream (\$STDIN).

To designate the data for the job or session, the user must precede the data with the :DATA command and terminate it with the :EOD command. The :DATA command implicitly initiates communication with MPE/3000 and thus is the only command that is not entered within a formally-initiated job or session. (Note that the :DATA command does not apply to data from non-auto recognizing devices, or data imbedded in the job or session input stream.)

The :DATA command format is

```
_:DATA [jobname,] username[/upass] .acctname[/apass] [:filename]
```

| | | |
|---------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------|
| (A complete job or session identification.) | <i>jobname</i> | The name of the job or session that is to read the data. (Optional parameter.) |
| | <i>username</i> | The user's name, as established in MPE/3000 by the user with Account Manager Capability. (Required parameter.) |
| | <i>upass</i> | The user password. (Required if user has a password.) |
| | <i>acctname</i> | The name of the account, as established by the user with System Manager Capability. (Required parameter.) |
| | <i>apass</i> | The account password. (Required if account has a password.) |
| <i>filename</i> | An additional qualifying name for the data that can be used by the job or session to access the data. It may be used, for example, to distinguish two separate data decks from different card readers read by the same program. If <i>filename</i> is omitted, no such distinguishing name is assigned. (Optional parameters.) | |

The data can *only* be read by a job or session that has the same identity:

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

The data exists in the system until read by the job/session.

If the *filename* parameter is omitted from the :DATA command, then the data can be read by *any* access from the job with the corresponding identity.

If the job attempts to read data but omits the file *formaldesignator* when opening the data file, any file preceded by a :DATA command referencing that job's identity will satisfy the request.

The *jobname*, *username*, *acctname*, and *filename* parameters are all names that can contain up to eight alphanumeric characters, beginning with a letter.

EXAMPLE:

To designate a card data file for a session identified as *JOBSP*, *BLACK.ACTSP*, the user submits the following *:DATA* command on a card preceding the data deck:

```
▶  :DATA JOBSP,BLACK.ACTSP; FILESP
    .
    .
    .
    (USER 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, as shown in Section V), and the filename *FILESP*.

PREMATURE JOB OR SESSION TERMINATION

A *job* is terminated before an *:EOJ* command is encountered, if any of the following events occur:

1. The Console Operator issues a command to terminate the job.
2. MPE/3000 aborts the job because an error occurred in the interpretation or execution of an MPE/3000 command.
3. MPE/3000 aborts the job because a subsystem error was encountered.
4. A program within the job is aborted.
5. A second *:JOB* command or a *:DATA* command is encountered. It will be executed, resulting in job termination (unless it is read as part of a program's data input from a file/device, in which case it signals an end-of-file to the program but is ignored by MPE/3000).
6. The central processor time limit specified in the *cpulimit* parameter of the *:JOB* command (or its default value) was exceeded.

Events 2 through 4 (above) place the job in an *error state* (by setting the error state flag). Normally, this results in job termination. But, if the user anticipates an error as a result of a specific command, he can override premature termination by using the *:CONTINUE* command before that command. The *:CONTINUE* command format is

:CONTINUE

The *:CONTINUE* command permits the job to continue even though the following command results in an error (in which case the error state indication is re-set).

EXAMPLE:

Suppose the user submits a job to execute three programs (with the :RUN command), but anticipates a probable terminating error in the first program. To continue the job and execute the second and third programs, he uses the :CONTINUE command as follows:

```
:JOB JNAM,UNAM.ACCTNAM  
:CONTINUE  
:RUN PROG1  
:RUN PROG2  
:RUN PROG3  
:EOJ
```

A session may be terminated before a :BYE command is encountered, if any of the following events occur:

1. The Console Operator issues a command to terminate the session.
2. A second :HELLO command, or a :JOB or :DATA command, is encountered in the session input stream. This will implicitly result in a :BYE command, terminating the current session.
3. The central processor time limit specified in the *cpulimit* parameter of the :HELLO command was exceeded.

INTRODUCING JOBS FOR SCHEDULING

A user can introduce a job, or sequence of jobs, to MPE/3000 for scheduling. Such jobs are completely independent of the originating job. The user supplies a file containing the job stream images with:

:STREAM [filedesignator] [,character]

filedesignator An input ASCII file whose record size is less than 256 bytes. The default is the job input device.

character Any ASCII graphic character. The default is an exclamation point (!).

:STREAM job streams are different from job streams originating on accepting devices, in that *character* is to be used in place of the standard ":" appearing in column 1; :STREAM replaces every *character* in column 1 with ":" as the stream is read.

This facility is enabled and disabled by the console operator who, when enabling, designates an input device to be associated with these job streams. The job stream specified by :STREAM, then, is indistinguishable from a real job stream originating from that device; e.g., the default output device for the job listing is the one associated with that (input) device (overridable with OUTCLASS = parameter of the :JOB command).

The command initiates processing by scanning for a legitimate JOB command, and proceeds by "spooling" each job onto disc and scheduling it for execution. This operation continues until physical end-of-file is detected, at which time control is returned immediately to the user. (Note that this is the "spooling" time; the jobs are actually executed later, independently of the requestor.) When this command is aborted (by pressing BREAK during a non-programmatic invocation) the current and all subsequent jobs are not processed; incomplete spooled disc space is returned.

As each :JOB command is encountered, it is analyzed for validity. Correct :JOB commands will cause the assigned job number to be returned to the originating user, when the job has been readied for job-dispatching; :JOB command errors will be immediately returned, without being printed on the operator's console.

(The :STREAM facility will also accept :DATA decks in a similar fashion.)

This command can be issued from jobs, from sessions, during break, and programmatically it is abortable. (Abortable commands are those which can be terminated by pressing BREAK).

SECTION IV

Compiling, Interpreting, Preparing, and Executing Programs

After a user initiates a batch job or session, he can enter commands to compile, prepare, and execute user programs or to access various MPE/3000 subsystems. Most of these commands require references to files used for input and output. For instance, a command to compile a program references a file that contains source program input, another used for program listing output, and a third used for object program output. Because of the importance of file references as command parameters, some of the rules for specifying files are introduced before the compilation, preparation, and execution commands themselves. The complete rules concerning files are discussed in Sections V and VI.

REFERENCING FILES

When compiling, preparing, or executing programs, the user can designate the files to be used for input and output in either of two ways:

1. By naming the files as positional parameters (called *actual file designators*) in the compilation, preparation, or execution command.
2. By omitting optional parameters from the compilation, preparation, or execution command, allowing MPE/3000 to assign standard default files.

Specifying Files as Command Parameters

The user can name the following types of files as parameters in a compilation, preparation, or execution command.

- System-Defined Files
- User Pre-Defined Files
- New Files
- Old Files

SYSTEM-DEFINED FILES. System-defined file designators indicate those files that MPE/3000 uniquely identifies as standard input/output devices for a job/session. They are referenced by the following actual file designators:

| Actual File Designator | Device/File Referenced |
|------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| \$STDIN | A filename indicating the standard job or session input device (that from which the job or session is initiated). For a job, this is typically a card reader. For a session, this is typically a terminal. Input data images in the \$STDIN file should not contain a colon in Column 1, since this indicates the end-of-data. (When data is to be delimited, this is properly done through the :EOD command, which executes no other functions.) |
| \$STDINX | Equivalent to \$STDIN, except that MPE/3000 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 :JOB, :DATA, :EOJ, and :EOD are exceptions that always indicate the end-of-data and are <i>never</i> read as data.) Furthermore, any requests with a byte count less than 5 indicate the end-of-data in this file, if there is a colon in Column 1. |
| \$STDLIST | A filename indicating the standard job or session listing device. The job or session listing device is customarily a printer for a batch job and a terminal for a session. |
| \$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/3000 but no physical output is actually performed. Thus, \$NULL can be used to discard unneeded output from a running program. |

USER PRE-DEFINED FILES. A user pre-defined file is any file that was previously defined or redefined in a :FILE command, as discussed in Section V. In other words, it is a back-reference to that :FILE command. In compilation, preparation, or execution commands, the actual file designator of this file is written as

**formaldesignator*

formaldesignator = The name used in the *formaldesignator* parameter of the :FILE command (Section V).

NEW FILES. New files are files that have not yet been created, and are being created/opened for the first time by the current batch job or interactive session. New files can have the following actual file designators:

| Actual File Designator | File Referenced |
|------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| \$NEWPASS | A temporary disc file that can be automatically passed to any succeeding MPE/3000 command within the same job/session, which references it by the filename \$OLDPASS. (Passing is explained in later examples.) Only one such file with this designation can exist in the job/session at any one time. (When \$NEWPASS is closed, its name is automatically changed to \$OLDPASS, and any previous file named \$OLDPASS in the job/session is deleted.) |
| <i>filereference</i> | Any other new file to which the user has access. Unless the user specifies otherwise, this is a temporary file, residing on disc, that is destroyed upon termination of the program. If closed as a <i>job/session temporary file</i> , as shown in Section V, it is saved until the end of the job/session, when it is purged. If closed as a permanent file, it is saved until purged by the user. Typically, this format consists of a file name containing up to eight alphanumeric characters, beginning with a letter. In addition, other elements (such as a group name, account name or lockword) can be specified. The complete rules governing the <i>filereference</i> format are presented in Section V. |

OLD FILES. Old files are existing files currently-resident in the system. They may be named by the following designators:

| Actual File Designator | File Referenced |
|------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| \$OLDPASS | The name of the temporary file last closed as \$NEWPASS. |
| <i>filereference</i> | Any other old file to which the user has access. It may be a job/session temporary file created in this or a previous program in the current job/session, a permanent file saved by any program in any job/session, or a permanent file built (with the :BUILD command) in any job/session. The format is the same as <i>filereference</i> , noted above and defined in Section V. |

INPUT/OUTPUT SETS. All of the preceding actual file designators can be classified as those used as input parameters (*Input Set*) and those used as output parameters (*Output Set*). These sets, referred to frequently throughout this manual, are defined as follows:

Input Set

| | |
|--------------------------|------------------------------------------------------------------------------------------|
| \$STDIN | The job/session input device. |
| \$STDINX | The job/session input device with commands allowed. |
| \$OLDPASS | The last \$NEWPASS file closed. |
| \$NULL | A constantly-empty file that will produce an end-of-file indication whenever it is read. |
| <i>*formaldesignator</i> | A back-reference to a previously-defined file. |
| <i>filereference</i> | A file name, and perhaps account and group names and a lockword. |

Output Set

| | |
|--------------------------|------------------------------------------------------------------|
| \$STDLIST | The job/session listing device. |
| \$OLDPASS | The last file passed. |
| \$NEWPASS | A new temporary file to be passed. |
| \$NULL | A constantly-empty file. |
| <i>*formaldesignator</i> | A back-reference to a previously-defined file. |
| <i>filereference</i> | A file name, and perhaps account and group names and a lockword. |

Further information on how these files are specified is presented under "Implicit :FILE Commands" in Section V.

Specifying Files By Default

When a user omits an optional file parameter from a compilation, preparation, or execution command, the subsystem invoked uses one of the members of the input or output set by default. The file designator assigned depends on the specific command, parameter, and operating mode, as noted later in this section.

USING THE BASIC/3000 INTERPRETER

The BASIC/3000 Interpreter is generally used for on-line programming during sessions. However, it can also be used to interpret BASIC/3000 programs submitted in the batch-job mode. In either case, the user calls the BASIC/3000 Interpreter with a command of the following format.

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

- commandfile* Source file (device) from which BASIC commands and statements are input. This can be any ASCII file from the input set. (The designator *BASCOM*, however, must *not* be used.) If omitted, the default file \$STDINX is assigned; (in sessions, this is typically the terminal; in batch jobs, it is usually the card reader). (Optional parameter.)
- inputfile* Source file containing data input to BASIC program. This can be any ASCII file from the input set. (The designator *BASIN*, however, must *not* be used.) If omitted, the default file \$STDINX is assigned; (for sessions, this is typically the terminal; in jobs, it is typically the card reader). (Optional parameter.)
- listfile* Destination file for BASIC program listing and output. Can be any ASCII file from output set. (The designator *BASLIST*, however, must *not* be used.) If omitted, the default value \$STDLIST (typically, the terminal for sessions or the printer for jobs) is assigned. (Optional parameter.)

EXAMPLES:

The following :BASIC command, entered during a session, specifies (by default) a terminal as the source for BASIC commands and input data, and the destination of output data generated by the user's program. (This is the application of the :BASIC command used most commonly.)

`:BASIC`

In special cases, the programmer may not use the terminal for input/output. The next :BASIC command, encountered during a batch job input through the card reader specifies the card reader (default input device) as the source of BASIC commands, the user-created disc file DATABANK as the source of input data, and the line printer (default list device) as the destination of output data. (Notice that the omission of the first parameter is indicated by beginning the parameter list with a comma.)

`:BASIC , DATABANK`

When the BASIC/3000 Interpreter has been entered, a greater-than sign (>) is used as the prompt character. This is returned by the terminal during sessions or entered by the user during jobs.

COMPILING/PREPARING/EXECUTING PROGRAMS

User source programs written in compiler languages undergo the operational steps outlined below. In most cases, the details of these steps will be invisible to the user during normal compilation and execution. When necessary, however, the user can advance through each of these steps independently, completely controlling the specifics of each event along the way.

1. The source language program is *compiled* (translated by a compiler) into binary form and stored as one or more relocatable binary modules (RBM's) in a specially-formatted disc file called a user subprogram library (USL). There is one RBM for each program unit. (A program unit is a self-contained set of statements that is the smallest divisible part of a program or subprogram. In FORTRAN/3000, this can be a main program, subroutine, function, or block data unit. In SPL/3000, it can be an outer block, or procedure. In COBOL/3000, it can be a main program, subroutine, or section.) In USL form, however, the program is not yet executable.
2. The USL is then *prepared* for execution by the MPE/3000 Segmenter. The Segmenter binds the RBM's from the USL into linked, re-entrant code segments organized on a program file. (In preparation, only *one* USL can be used for RBM input to the program file.) At this point, the special segment for the input of user data (the stack) is also initially defined.
3. The program file is *allocated and executed*. In allocation, the segments from the program file are bound to referenced external segments from segmented libraries (SL's). Then the first code segment to be executed, and the stack are moved into main memory and execution begins.

Over a period of time, the user can produce RBM's on the same USL through several compilations. If there is only one main program or outer block RBM containing active entry points, this USL can be prepared onto a program file and run. This allows the user to compile his main program and procedures separately.

During compilation, if a program unit is designated as privileged, the corresponding RBM is flagged as privileged in the USL.

During preparation, the USL and program files are opened as job temporary files. For both files, the Segmenter first searches for a file of the proper name that exists as a 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 job temporary file domain, and prepares into this. If any RBM from the USL is flagged as privileged, the user must have the privileged mode (PM) optional capability to 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 also determined (by the user or by the default option).

A particular program can be run by many user processes at the same time, because code in a program is inherently sharable.

Before allocation/execution is initiated, MPE/3000 checks to verify the following points:

- a. 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.
- b. For program files that are temporary files in the job/session domain, the capabilities assigned to the program must not exceed those of the user running the program; otherwise, an error occurs.
- c. 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 the following pages, all of these commands are described.

Compilation Only

To compile a source-language program, the user enters a command of the following format:

```
_:compiler [textfile] [, [uslfile] [, [listfile] [, [masterfile] [, newfile] ] ] ] ] ]
```

| | |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>compiler</i> | FORTRAN for the FORTRAN/3000 compiler; SPL for the SPL/3000 compiler; COBOL for the COBOL/3000 compiler; RPG for the RPG/3000 compiler; or BASICOMP for the BASIC/3000 compiler. |
| <i>textfile</i> | Input file (device) from which the source program is to be read. This can be any ASCII file from the input set. (The following designators, however, must <i>not</i> be used: <i>SPLTEXT</i> (SPL/3000), <i>FTNTEXT</i> (FORTRAN/3000), <i>COBTEXT</i> (COBOL/3000), <i>RPGTEXT</i> (RPG/3000), or <i>BSCTEXT</i> (BASIC/3000 compiler).) If omitted, the default file \$STDIN (the current input device) is used. (Optional parameter.) |
| <i>uslfile</i> | The name of the USL file on which the object program is written. This can be any binary file from the output set. (The following designators, however, must <i>not</i> be used: <i>SPLUSL</i> (SPL/3000), <i>FTNUSL</i> (FORTRAN/3000), <i>COBUSL</i> (COBOL/3000), <i>RPGUSL</i> (RPG/3000), or <i>BSCUSL</i> (BASIC/3000 compiler).) If the <i>uslfile</i> parameter is entered, it must indicate a file previously created by the user in one of three ways: <ol style="list-style-type: none">1. By saving a USL file (through the <i>:SAVE</i> command, discussed in Section V) created by a previous compilation where the default value was used for the <i>uslfile</i> parameter. |

2. By building the USL (through the segmenter subsystem command — BUILDUSL, discussed in Section VII.)
3. By creating a new file of USL type (through the :BUILD command discussed in Section V, with the decimal code of 1024 or the mnemonic *USL* used for the *filecode* parameter).

If the *uslfile* parameter is omitted, the default file \$OLDPASS is assigned, unless there is no passed file (in which case, \$NEWPASS is used). (Optional parameter.)

| | |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>listfile</i> | The name of the file to which the program listing is to be generated. This can be any ASCII file from the output set. (The following designators, however, must <i>not</i> be used: <i>SPLLIST</i> (SPL/3000), <i>FTNLIST</i> (FORTRAN/3000), <i>COBLIST</i> (COBOL/3000), <i>RPGLIST</i> (RPG/3000), or <i>BSCLIST</i> (BASIC/3000 compiler).) If omitted, the default file \$STDLIST (typically the terminal in a session or the printer in a batch job) is assigned. (Optional parameter.) |
| <i>masterfile</i> | The name of a file to be optionally merged with <i>textfile</i> and written onto a file named <i>newfile</i> , as discussed in the manuals covering compilers. (The following designators, however, must <i>not</i> be used: <i>SPLMAST</i> (SPL/3000), <i>FTNMAST</i> (FORTRAN/3000), <i>COBMAST</i> (COBOL/3000), or <i>RPGMAST</i> (RPG/3000).) If <i>masterfile</i> is omitted, no merging takes place. Not available for BASIC/3000 compiler. (Optional parameter.) |
| <i>newfile</i> | The file on which the (re-sequenced) records from <i>textfile</i> (and <i>masterfile</i>) are placed. (The following designators, however, must <i>not</i> be used: <i>SPLNEW</i> (SPL/3000), <i>FTNNEW</i> (FORTRAN/3000), <i>COBNEW</i> (COBOL/3000), or <i>RPGNEW</i> (RPG/3000).) When <i>newfile</i> is omitted, no <i>newfile</i> is created. Not available for BASIC/3000 compiler. (Optional parameter.) |

EXAMPLES:

The following command compiles a FORTRAN/3000 source program entered by a user through the job/session input device, into an object program in the USL file \$NEWPASS. It also transmits listing output to the job/session list device. (If the next command is one to prepare an object program, \$NEWPASS can be passed to it as an input file. Note that a file can only be passed between commands or programs within the same job or session.)

:FORTRAN

The following example illustrates the passing of a file between two successive compilations. In this example, the `:FORTRAN` command compiles a program into the USL file `$NEWPASS` (because the default value for `uslfile` is taken, and no passed file presently exists). The `:SPL` command compiles another program into the same USL, passed to this command under the redesignation `$OLDPASS`. (When no passed file exists, `$NEWPASS` is used; when a passed file exists, that file (`$OLDPASS`) is used.) After the second compilation, `$OLDPASS` contains RBM's from both compilations:

```

:JOB MYNAME.MYACCT
:FORTRAN

      (FORTRAN SOURCE PROGRAM)
      .
      .
      .
:EOJ
:SPL

      .
      .
      .
      (SPL SOURCE PROGRAM)
:EOJ

      .
      .
      .
:EOJ

```

The following command compiles a COBOL/3000 source 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`.

```

:COBOL SOURCE, OBJECT, LISTFL

```

Compilation/Preparation

To compile and prepare for execution a source-language program, the user enters a command in the following format. (The USL file created during compilation is a temporary file passed directly to the preparation mechanism; it is accessible to the user as `$OLDPASS` only if the `progfile` is not `$NEWPASS`.)

```

:compiler [textfile] [, [progfile] [, [listfile] [, [masterfile] [, newfile] ]]]

```

compiler FORTPREP for the FORTRAN/3000 compiler; SPLPREP for the SPL/3000 compiler; COBOLPREP for the COBOL/3000 compiler; RPGPREP for the RPG/3000 compiler; or BASICPREP for the BASIC/3000 compiler.

textfile Input file (device) from which the source program is read. This can be any ASCII file from the input set. (The following designators, however, must *not* be used: `SPLTEXT` (SPL/3000), `FTNTEXT` (FORTRAN/3000), `COBTEXT` (COBOL/3000), `RPGTEXT` (RPG/3000), or `BSCTEXT` (BASIC/3000 compiler).) If omitted, the default file `$STDIN` (the current input device) is used. (Optional parameter.)

progfile

The name of the program file onto which the prepared program segments are to be written. This can be any binary file from the output set. The user must create this program file (unless the default \$NEWPASS is taken). He does this in one of two ways:

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

NOTE: Code segments in a program file cannot lie across disc extent boundaries. Thus, all code segments in such files must be constructed within one extent.

2. By specifying a non-existent file in the *progfile* parameter of the :FORTPREP, :SPLPREP, :COBOLPREP, :RPGPREP, or :BASICPREP, in which case a job-temporary file of the correct size and type is created.

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

listfile

The name of the file to which the program listing is written. This can be any ASCII file from the output set. (The following designators, however, must *not* be used: *SPLLIST* (SPL/3000), *FTNLIST* (FORTRAN/3000), *COBLIST* (COBOL/3000), *RPGLIST* (RPG/3000), or *BSCLIST* (BASIC/3000 compiler).) If omitted, the default file assigned is \$STDLIST (usually the terminal in a session or the printer in a batch job). (Optional parameter.)

masterfile
newfile }

The same meanings described under *compilation*.

NOTE: This command is equivalent to a compilation command followed by a :PREP command (as described later in this section). Thus, for FORTRAN/3000, the command

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

is equivalent to

```
:FORTRAN [textfile]  
            , $NEWPASS  
            , listfile  
            , masterfile  
            , newfile  
  
:PREP     $OLDPASS  
            , progfile
```

EXAMPLES:

The following command compiles and prepares an SPL/3000 program entered through the job/session input device. The resulting program file is named \$NEWPASS, and the program listing is printed on the job/session list device. (If the next command is one to execute the program, the file \$NEWPASS is referenced in the execute command under the name \$OLDPASS.)

:SPLPREP

The next command compiles and prepares a FORTRAN/3000 source program input from a source file named SFILE into a program file named \$NEWPASS. The resulting program listing is printed on the job/session listing device.

:FORTPREP SFILE

Compilation/Preparation/Execution

To compile, prepare, and execute a program, a command of the following format is entered. (This command creates a temporary USL file that is not accessible by the user, and a program file available to the user as \$OLDPASS.)

:compiler [textfile] [, [listfile] [, [masterfile] [, newfile]]]

compiler FORTGO for the FORTRAN/3000 compiler; SPLGO for the SPL/3000 compiler; COBOLGO for the COBOL/3000 compiler; RPGGO for the RPG/3000 compiler, and BASICGO for the BASIC/3000 compiler.

textfile Input file (device) from which source program is read. This can be any ASCII file from the input set. (The following designators, however, must *not* be used: *SPLTEXT* (SPL/3000), *FTNTEXT* (FORTRAN/3000), *COBTEXT* (COBOL/3000), *RPGTEXT* (RPG/3000), or *BSCTEXT* (BASIC/3000 compiler).) If omitted, the default file \$STDIN (the current input device) is assigned. (Optional parameter.)

listfile The name of the file to which the program listing is transmitted. This can be any ASCII file from the output set. (The following designators, however, must *not* be used: *SPLLIST* (SPL/3000), *FTNLIST* (FORTRAN/3000), *COBLIST* (COBOL/3000), *RPGLIST* (RPG/3000), or *BSCLIST* (BASIC/3000 compiler).) If omitted, the default file assigned is \$STDLIST (normally the terminal for sessions or printer for batch jobs). (Optional parameter.)

masterfile }
newfile } The same meanings described under *compilation*.

NOTE: This command is equivalent to a compilation command, followed by a :PREP command, followed by a :RUN command (as described later in this section). Thus, for FORTRAN/3000, the command

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

is equivalent to

```
_:FORTRAN textfile,
           $NEWPASS
           ,listfile
           ,masterfile
           ,newfile

_:PREP $OLDPASS, $NEWPASS

_:RUN $OLDPASS
```

EXAMPLES:

The command shown below compiles, prepares, and executes a COBOL/3000 program entered through the job/session input device. The program listing is printed on the job/session list device.

```
_:COBOLGO
```

The following command compiles, prepares, and executes an SPL/3000 program residing in the disc file SOURCE and transmits the program listing to the job/session list device.

```
_:SPLGO SOURCE
```

Preparation Only

If a user's source program has been compiled onto a USL file, he can prepare it for execution with the :PREP command. (When writing this command, recall that keyword parameters can appear in any order after the positional parameters.)

```
_:PREP uslfile,progfile

[;ZERODB]
[;PMAP]
[;MAXDATA=segsize]
[;STACK=stacksize]
[;DL=dlsize]
[;CAP=caplist]
[;RL=filename]
```

| | |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>uslfile</i> | The name of the USL file (from the input set) on which the program has been compiled. (Required parameter.) |
| <i>progfile</i> | The name of the program file onto which the prepared program segments are to be written. This can be any binary file from the output set. The user must create this program file. He does this in one of two ways: <ol style="list-style-type: none"> 1. By creating a new file of program-file type (through the :BUILD command discussed in Section V, with the decimal code of 1029 or the mnemonic <i>PROG</i> used for the <i>filecode</i> parameter). <p><i>NOTE: Code segments in a program file cannot lie across disc extent boundaries. Thus, all code segments in such files must be constructed within one extent.</i></p> <ol style="list-style-type: none"> 2. By specifying a non-existent file in the <i>progfile</i> parameter of the :PREP command (or in the <i>progfile</i> parameter of the segmenter subsystem command — PREPARE, discussed in Section VII), in which case a file of the correct size and type is created. This file is a JOB temporary file. (Required parameter.) |
| <i>ZERODB</i> | An indication that the initially-defined user-managed (DL-DB) area, and uninitialized portions of the DB-Q (initial) area, will be initialized to zero. If this parameter is omitted, these areas are not affected. (Optional parameter.) |
| <i>PMAP</i> | An indication that a descriptive listing of the prepared program will be produced on the file whose formal designator is SEGLIST; if no :FILE command referencing SEGLIST is encountered, the listing is produced on the job/session listing device. If this parameter is omitted, the listing is not produced. (Optional parameter.) |
| <i>segsiz</i> | Maximum stack area (Z-DL) size permitted, in <i>words</i> . This parameter is included if the user expects to change the size of the DL-DB or Z-DB areas during process execution. If omitted, MPE/3000 assumes that he will not change these areas. (Optional parameter.) |
| <i>stacksiz</i> | The size of the user's initial local data area (Z-Q (initial)) in the stack, in <i>words</i> . (This value <i>must</i> exceed 511 words.) This overrides the <i>stacksiz</i> estimated by the segmenter, which applies if the <i>stacksiz</i> parameter is omitted. (The default is a function of estimated stack requirements for each program unit in the program. Since it is difficult for the system to predict the behavior of the stack at run time, the user may want to override the default by supplying his own estimate with <i>stacksiz</i> .) (Optional parameter.) |
| <i>dlsiz</i> | The DL-DB area to be initially assigned to the stack. This area is mainly of interest only in programmatic applications, and is discussed in detail in Section II. If the <i>dlsiz</i> parameter is omitted, a value estimated by the segmenter applies. (Optional parameter.) |

caplist

The *capability-class attributes* associated with the user's 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 | | |

The user who issues the :PREP command can only specify capabilities that he himself possesses (through assignment by the System or Account Manager). If the user does not specify any capabilities, the IA and BA capabilities (if possessed by the user) will be assigned to this program. (Optional parameter.)

filename

The name of a relocatable procedure library (RL) file to be searched to satisfy external references during preparation as defined in Section VII. This can be any binary permanent file of type RL, as described in Section VII. It need not belong to the 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.)

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

EXAMPLES:

The following command prepares a program from the USL file named USEFILE and stores it in a program file named PROGFILE. The optional parameters will be those assigned by default. The program's capability-class attributes will be the standard capabilities of the preparing user.

:PREP USEFILE, PROGFILE

The next command will accomplish the same function as the previous command, except that the prepared program will be listed, a stacksize of 500 words will be established, and the program will be assigned only the batch-access capability.

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

An example of the listing of the prepared program, requested by the PMAP parameter of the :PREP (and :PREPRUN) command, is shown in Figure 4-1. On this listing, significant entries are indicated with arrows, keyed to the discussion below.

NOTE: All numbers in the listing except Item 13 (elapsed time) and Item 20 (central processor time) are octal values.

| 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. |

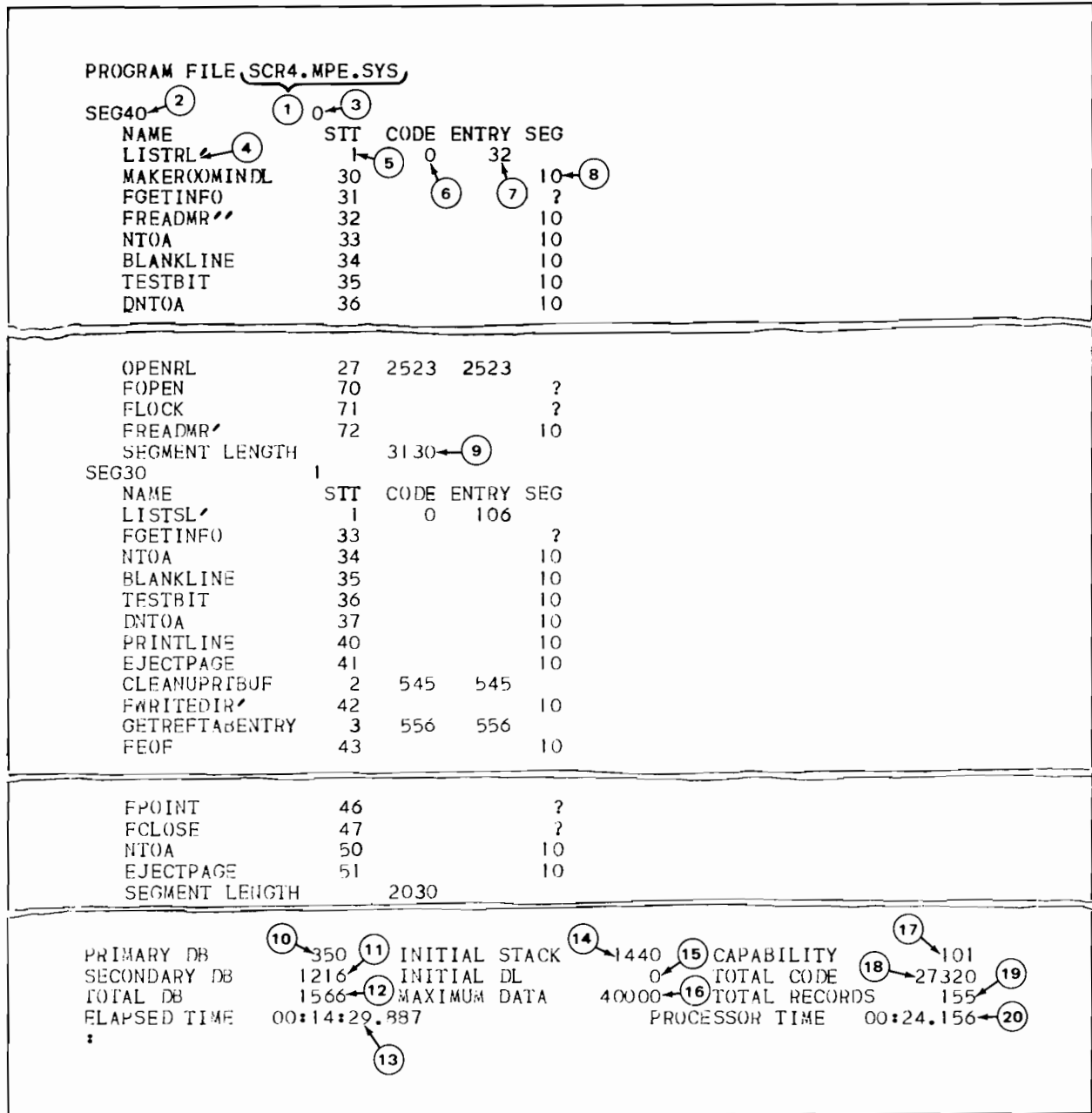


Figure 4-1. Listing of Prepared Program

Preparation/Execution

Programs compiled in USL files can be prepared *and* executed with the `:PREPRUN` command. This 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`. The program file is available to the user as `$OLDPASS`.) The command format is:

```
_:PREPRUN uslfile[,entrypoint]
```

```
[:NOPRIV]
[:PMAP]
[:DEBUG]
[:LMAP]
[:ZERODB]
[:MAXDATA=segsz]
[:PARM=parameternum]
[:STACK=stacksize]
[:DL=dlsz]
[:LIB=library]
[:CAP=caplist]
[:RL=filename]
```



- | | |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>uslfile</i> | The name of the USL file (from the input set) on which the program has been compiled. (Required parameter.) |
| <i>entrypoint</i> | The program entry-point where execution is to begin. This may be the primary entry point of the program, or any secondary entry point in the program's outer block. If the parameter is omitted, the primary entry point is assigned by default. (Optional parameter.) |
| <i>NOPRIV</i> | A declaration that the program segments will be placed in <i>non-privileged</i> (user) mode. This parameter is intended for programs prepared with the privileged-mode capability. Normally, program segments containing privileged instructions are executed (run) in privileged mode <i>only</i> if the program was <i>prepared</i> with the 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 <i>NOPRIV</i> is specified in the <code>:PREPRUN</code> 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.) |
| <i>PMP</i> | An indication that a listing describing the <i>prepared</i> program will be produced on the file whose formal designator is <code>SEGLIST</code> ; if no <code>:FILE</code> command referencing <code>SEGLIST</code> is encountered, the listing is produced on the job/session listing device. If this parameter is omitted, the listing is not produced. (Optional parameter.) |
| <i>DEBUG</i> | An indication that a breakpoint will be set on the first executable instruction of the program. |

| | |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>LMAP</i> | An indication that a descriptive listing of the <i>allocated</i> (loaded) program will be produced on the file whose formal designator is LOADLIST; if no :FILE command referencing LOADLIST is encountered, the listing is produced on the job/session listing device. If LMAP is omitted, the listing is not produced. (Optional parameter.) |
| <i>ZERODB</i> | An indication that the initially-defined user-managed (DL-DB) area, and uninitialized portions of the DB-Q (initial) area, will be initialized to zero. If this parameter is omitted, these areas are not affected. (Optional parameter.) |
| <i>segsiz</i> | Maximum stack area (Z-DL) size permitted. This parameter is included if the user expects to change the size of the DL-DB or Z-DB areas during process execution. If omitted, MPE/3000 assumes that he will not change these areas. (Optional parameter.) |
| <i>parameternum</i> | A value that can be passed to the user's program as a general parameter for control or other purposes; when the program is executed, this value can be retrieved from the address Q (initial)-4, where Q (initial) is the Q-address for the outer block of the program. The value can be an octal number or a signed or unsigned decimal number. If <i>parameternum</i> is omitted, the Q (initial)-4 address is filled with zeros. (Optional parameter.) |
| <i>stacksize</i> | The initial size of the user's initial local data area (Z-Q (initial)) in the stack, in words, as described in the discussion of the :PREP command. (This value must exceed 511 words.) (Optional parameter.) |
| <i>dlsiz</i> | The initial DL-DB area size, as described in the discussion of the :PREP command. (Optional parameter.) |
| <i>library</i> | The order in which applicable segmented procedure libraries are searched to satisfy external references during allocation, where: <p style="margin-left: 40px;">G = Group library, followed by account public library, followed by system library. (These libraries are defined in Section VII.)</p> <p style="margin-left: 40px;">P = Account public library, followed by system library.</p> <p style="margin-left: 40px;">S = System library.</p> <p>If no parameter is specified, S is assumed. (Optional Parameter.)</p> |
| <i>caplist</i> | The capability-class attributes associated with the user's program, as described in the discussion of the :PREP command. (Optional parameter.) |
| <i>filename</i> | The name of a relocatable procedure library (RL) file to be searched to satisfy external references during preparation, as described in the discussion of the :PREP command. (Optional parameter.) |

For *stacksize*, *dlsiz*, and *maxdata*, a value of -1 indicates the default value; it is equivalent to omitting the parameter.

EXAMPLES:

The following command prepares and executes a program on the USL file USEF, with no special parameters declared. (All default values apply.)

:PREPRUN USEF

The next command prepares and executes 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:

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

An example of the listing of the loaded program, requested by the LMAP parameter of the :PREPRUN (and :RUN) command, is shown in Figure 4-2. This listing shows the externals referenced by the program and the segments from segmented libraries to which they were bound. On this listing, significant entries are indicated with arrows, keyed to the discussion below.

| 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. |

PROGRAM FILE SCR4.MPE.SYS

| | (1) | (4) | (5) | (6) | (7) | (8) | (9) | (10) |
|-------------|--------|-----|-----|-----|-----|-----|-----|------|
| TERMINATE | PROG 0 | 50 | 11 | SSL | 0 | 2 | 37 | |
| SENDMAIL | PROG 0 | 46 | 11 | SSL | 0 | 2 | 40 | |
| DEBUG | PROG 0 | 44 | 11 | SSL | 0 | 1 | 52 | |
| RECEIVEMAIL | PROG 0 | 6 | 11 | SSL | 0 | 1 | 40 | |
| AWAKE | PROG 0 | 5 | 11 | SSL | 0 | 4 | 33 | |
| WHO | PROG 0 | 4 | 11 | SSL | 0 | 4 | 45 | |
| FREADDIR | PROG 0 | 45 | 10 | SSL | 0 | 1 | 0 | |
| FWRITEDIR | PROG 0 | 44 | 10 | SSL | 0 | 2 | 0 | |
| FWRITE | PROG 0 | 43 | 10 | SSL | 0 | 3 | 0 | |
| DLSIZE | PROG 0 | 41 | 10 | SSL | 0 | 15 | 42 | |
| PRINT | PROG 0 | 15 | 7 | SSL | 0 | 11 | 45 | |
| SETJCW | PROG 0 | 13 | 7 | SSL | 0 | 1 | 45 | |
| GETJCW | PROG 0 | 12 | 7 | SSL | 0 | 2 | 45 | |
| ADJUSTUSLF | PROG 0 | 42 | 6 | SSL | 0 | 11 | 46 | |
| FPOINT | PROG 0 | 46 | 2 | SSL | 0 | 5 | 1 | |
| RESETDB | PROG 0 | 34 | 4 | SSL | 0 | 7 | 33 | |
| | | 26 | 2 | | | | | |
| SETSYSDB | PROG 0 | 33 | 4 | SSL | 0 | 11 | 33 | |
| | | 25 | 2 | | | | | |
| FCONTROL | PROG 0 | 14 | 2 | SSL | 0 | 3 | 1 | |
| PROCTIME | PROG 0 | 4 | 2 | SSL | 0 | 15 | 44 | |
| TIMER | PROG 0 | 3 | 2 | SSL | 0 | 23 | 33 | |
| GETUSERMODE | PROG 0 | 10 | 11 | SSL | 0 | 4 | 44 | |
| | | 35 | 4 | | | | | |
| | | 27 | 2 | | | | | |
| | | 55 | 1 | | | | | |
| LOADEDLSSEG | PROG 0 | 53 | 1 | SSL | 0 | 2 | 47 | |
| GETPRIVMODE | PROG 0 | 45 | 11 | SSL | 0 | 5 | 44 | |
| | | 32 | 4 | | | | | |
| | | 24 | 2 | | | | | |
| | | 52 | 1 | | | | | |
| QUIT | PROG 0 | 7 | 11 | SSL | 0 | 11 | 31 | |
| | | 52 | 10 | | | | | |
| | | 25 | 5 | | | | | |
| | | 50 | 1 | | | | | |

| | | | | | | | |
|----------|--------|----|----|-----|---|---|---|
| FGETINFO | PROG 0 | 14 | 11 | SSL | 0 | 4 | 2 |
| | | 46 | 10 | | | | |
| | | 32 | 6 | | | | |
| | | 56 | 3 | | | | |
| | | 15 | 2 | | | | |
| | | 33 | 1 | | | | |
| | | 31 | 0 | | | | |

131 132 133 134 135 155 156 177 200 201

Figure 4-2. Listing of Loaded Program

Execution

Programs that have been compiled and prepared, and that therefore exist on program files, can be executed by the `:RUN` command, entered as follows:

```
:RUN progfile[,entrypoint]
    [;NOPRIV]
    [;LMAP]
    [;DEBUG]
    [;MAXDATA=segsiz]
    [;PARM=parameternum]
    [;STACK=stacksize]
    [;DL=dlsiz]
    [;LIB=library]
```

- | | |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>progfile</i> | The name of the program file (from the input set) that contains the prepared program. (Required parameter.) |
| <i>entrypoint</i> | The program entry-point where execution is to begin. This may be the primary entry-point of the program or any secondary entry-point in the program's outer block. If this parameter is omitted, the primary entry-point (where (execution normally begins) is assigned by default. (Optional parameter.) |
| <i>NOPRIV</i> | A declaration that the program segments will be placed in <i>non-privileged</i> mode; this parameter is intended for programs prepared with the privileged-mode capability. Normally, program segments containing privileged instructions are executed (run) in privileged mode <i>only</i> if the program was <i>prepared</i> with the 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 <i>NOPRIV</i> is specified in the <code>:RUN</code> command, all program segments are placed in the <i>non-privileged</i> mode. (Library segments are not affected since their mode is determined independently.) (Optional parameter.) |
| <i>LMAP</i> | An indication that a listing of the <i>allocated</i> (loaded) program will be produced on the file whose formal designator is <code>LOADLIST</code> ; if no <code>:FILE</code> command referencing <code>LOADLIST</code> is encountered, the listing is produced on the job/session listing device. If <i>LMAP</i> is omitted, the listing is not produced. (Optional parameter.) |
| <i>DEBUG</i> | An indication that a breakpoint will be set on the first executable instruction of the program. This parameter is ignored when a non-privileged user runs a program having privileged mode capability. The parameter is also ignored if the user does not have read and write access to the program file. |
| <i>segsiz</i> | Maximum stack area (Z-DL) size permitted, as described in the discussion of the <code>:PREP</code> command. (Optional parameter.) |

| | |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>parameternum</i> | A decimal or octal value that can be passed to the user's program as a general parameter for control or other purposes; when the program is executed, this value can be retrieved from the address Q (initial) -4, where Q (initial) is the Q-address for the outer block of the program. The value can be an octal number or a signed or unsigned decimal number. If <i>parameternum</i> is omitted, the Q (initial) -4 address is filled with zeros. (Optional parameter.) |
| <i>stacksize</i> | The initial size of the user's initial local data area (Z-Q (initial)) in the stack, in words, as described in the discussion of the :PREP command. (This value must exceed 511 words.) (Optional parameter.) |
| <i>dlsiz</i> e | The initial DL-DB area size, as described in the discussion of the :PREP command. (Optional parameter.) |
| <i>library</i> | The order in which applicable segmented procedure libraries are searched to satisfy external references during allocation, where: <ul style="list-style-type: none"> G = Group library, followed by account public library, followed by system library. (These libraries are defined in Section VII.) P = Account public library, followed by system library. S = System library. <p>If no parameter is specified, <u>S</u> is assumed. (Optional parameter.)</p> |

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.

EXAMPLES:

The following command executes a program existing on the program file PROG3, with no special parameters specified. (All default values are used.)

```
_:RUN PROG3
```

The next command executes a program existing on the program file PROG4, beginning at the entry-point SECLAB. All segments in the program will run in non-privileged mode.

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

CALLING MPE/3000 SUBSYSTEMS

In addition to the BASIC/3000 interpreter and the SPL/3000, COBOL/3000, and FORTRAN/3000 compilers, various other programs are available that run as subsystems of MPE/3000. These programs, and the way in which they are accessed, are described below.

EDIT/3000

EDIT/3000, described in the *EDIT/3000* manual, is used to create and modify files of upper and lower-case ASCII characters. To call EDIT/3000, the user enters:

_:EDITOR [listfile]

listfile The name of a file that receives any output resulting from the EDIT/3000 command LIST specifying the OFFLINE option. (The designator *EDTLIST* must not be used.) If *listfile* is omitted, such output is transmitted to the job/session list device (*\$STDLIST*). (The file to be edited is specified after EDIT/3000 has been called.) (Optional parameter.)

STAR/3000

The Statistical Analysis Routines (STAR/3000), described in the *STAR/3000* manual, allows the user to access the statistical functions of the HP 3000 Scientific Library. STAR/3000 is called by this command:

_:STAR [listfile] [,NOLIST]

listfile File on which output from STAR/3000 is to be written. (Its formal designator is *STRLIST*.) If omitted, *\$STDLIST* (typically, the terminal for sessions and the printer for jobs) is used. (All commands are input from the job/session input device and results are output to *listfile*.) (Optional parameter.)

NOLIST A specification that STAR/3000 command input will *not* be listed on *listfile* when STAR/3000 is run in batch mode. (Optional parameter.)

TRACE/3000

TRACE/3000 allows the user to monitor program execution, checking the status of the program whenever a variable is changed or a label is passed. TRACE/3000 is invoked through compiler subsystem command records, as noted in the *TRACE/3000* manual.

SORT/3000

SORT/3000, described in the *SORT/3000* manual, allows the user to sort records in a file into prescribed order, or to merge pre-sorted records in multiple files into one sorted file. SORT/3000 can be invoked directly through MPE/3000 commands, or called, in phases, from user programs or procedures.

When invoking SORT/3000 through MPE/3000, the user first specifies the appropriate input/output files through MPE/3000 *:FILE* commands, or allows the default file designators to

take effect (as described in the *SORT/3000* manual.) He then invokes either the SORT program or the Merge program by entering one of the following commands.

For Sort:

```
_:RUN SORT [;STACK = size]
```

For Merge:

```
_:RUN MERGE [;STACK = size]
```

size The amount of main-memory storage to be used by the Sort or Merge program, in words. If *size* is omitted, the default value assigned is 1000 words. (Optional parameter.)

SDM/3000

The execution of on-line diagnostic programs, run by a person with the *Diagnostician* user attribute, is controlled by the HP/3000 Diagnostic Monitor (SDM/3000), described in *HP 3000 System Diagnostic Monitor*. This program allows the operator to invoke, execute, and modify diagnostic programs. It is called by entering:

```
_:RUN SDM
```

HP/3000 Stand-Alone Diagnostic Utility Program

Magnetic tapes containing stand-alone diagnostic programs are prepared through the HP/3000 Stand-alone Diagnostic Utility Program, described in *HP 3000 Stand-alone Diagnostic Utility Program*. These programs can then be loaded and executed without any other supporting software. The Utility Program is invoked through this command:

```
_:RUN SDUP; NOPRIV
```

MPE/3000 Segmenter

The MPE/3000 Segmenter can be accessed directly by the user, allowing him to enter commands that

- Create, delete, activate, and deactivate RBM's within a USL.
- Manage procedure libraries used to resolve external program references

The user accesses the segmenter by entering:

```
_:SEGMENTER [listfile]
```

listfile An ASCII file from the output set, to which any listable output is written. (The formal file designator is *SEGLIST*.) If *listfile* is omitted, the standard job/session list device (*\$STDLIST*) is assumed. (Optional parameter.)

He then enters commands relating to the USL's or procedure libraries. These commands are explained in Section VII:

2780/3780 Emulator

The Hewlett-Packard 2780/3780 Emulator permits transfer of data between an HP 3000 Computer 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. Control is passed to the Emulator by the following command:

```
:RJE [commandfile] [, [inputfile] [, [listfile] [, punchfile]]]
```

commandfile The file from which the Emulator reads its directives.

inputfile The file from which the Emulator reads input data to be transmitted to the remote computer.

listfile The file to which list output received from the remote computer is to be routed.

punchfile The file to which punched output received from the remote computer is to be routed.

SAMPLE PROGRAMS

The following sample programs illustrate, in context, some of the commands introduced in this and the previous section.

The following program shows the listing from a session where the programmer accessed and used the BASIC/3000 Interpreter:

```
:HELLO MATH,BOBBIE.ACMATH, GPMATH  
USER PASSWORD?  
NNNNNNNN  
SESSION NUMBER=#S231  
THU,MARCH 23, 1972, 2:35 PM  
HP32000B.02.08  
:BASIC  
>  
. (BASIC COMMANDS AND STATEMENTS)  
. .  
. .  
. .  
>EXIT  
:BYE  
CPU(SEC)=14  
CONNECT(MIN)=2  
THU,MARCH 23, 1972, 2:37 PM  
END OF SESSION
```

} (LOG-ON)

(COMMAND TO ACCESS BASIC)
(BASIC PROMPT CHARACTER,
FOLLOWED BY A BASIC COMMAND)

} (EXIT FROM BASIC)

} (LOG-OFF)



SECTION V

Managing Files

MPE/3000 organizes and handles as files all information input to and output from the computer, relieving the user of the software-writing effort normally involved when interacting directly with physical peripheral devices. Input and output are generally performed by sharable system code that translates file names specified by the user into actual hardware addresses, allowing the user to remain virtually unaware of detailed file specifications and hardware characteristics.

FILE CHARACTERISTICS

A file can contain MPE/3000 commands, programs, or data, or any combination of these elements, written in ASCII or binary code. For example, a file input from the card reader might contain MPE/3000 commands, a user program, and user data, all in ASCII code. Compilation of the user program would produce a USL file containing object output in binary code and another file containing a listable copy of the user's source program in ASCII code. Preparation of the object program produces a binary-coded program file containing the user's program in segmented form.

Within a file, information is organized as a set of logical records — fields of data input, processed, and output as a unit. A logical record is the smallest data grouping directly addressable by the user. Its length is specified by the user when he creates or defines the file.

Information is moved between a file and main memory in *physical records*; a physical record is the basic unit that can be transferred to or from the device on which the file resides. Physical records can be longer, shorter, or the same size as the logical records in the file. In files on disc or magnetic tape, physical records are organized as *blocks* that always contain an integral number of logical records; thus, on these devices, physical records are either longer than, or the same size as, logical records. (The block size is specified implicitly by the user in the command or intrinsic call that creates the file. But input/output and blocking are performed automatically by MPE/3000, freeing the user of responsibility for the actual record-handling details.)

On unit-record devices, however, the size of the physical records in a file is determined by the device itself. Thus, each physical record read from a card reader consists of one punched card; each physical record written to a line printer consists of one line of print. On unit record-devices in multi-record mode, *logical* records are *not* blocked. For example, on punched cards, each logical record is assumed to begin at the first column of the card; a single 100-character logical record is read as 80 characters from one card (physical record) and 20 characters from the next card. The next logical record is assumed to begin in the first column of the third card encountered. Similarly, when a file is transmitted to a printer, each logical record appears as one line of print (physical record), left-justified; if the logical record is longer than the print line, the remaining information is continued on the next line, also left-justified.

On disc, files can be organized to permit either sequential or direct access. Direct-access files contain logical records of fixed or undefined length. But sequential-access files contain logical records of fixed or varying length.

MPE/3000 manages each file on disc as a set of *extents*; each extent is an integral number of contiguously-located disc sectors. All extents (except possibly the last) are of equal size. When a file is opened, the first extent (containing at least one sector for file label information) is allocated immediately; other extents, up to a maximum of 16, are allocated as needed. Alternatively, the user can request immediate allocation of more than one extent when the file is opened. (The size of each extent is determined as noted in the discussion of the :FILE command parameter *numextents*, later in this section. All extents are allocated on the same disc drive.)

When a file contains only fixed length records, the user can calculate the effective disc space (in sectors) required by the file, with the following formula. (This formula applies to every type of disc drive supported by MPE/3000. It does not include disc space required by the file label.)

$$S = \left\lceil \frac{N}{B} \right\rceil \times \left\lceil \frac{LxB}{128} \right\rceil$$

- S* The number of disc sectors required by the file.
- N* The number of fixed-length logical records in the file.
- B* The number of logical records in a block (blocking factor).
- L* The length of each logical record, in 16-bit *words*.

The constant 128 is the number of words 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.)

EXAMPLE:

The effective disc space for a file containing 100 records of 50 words each, with a blocking factor of 3, would be calculated as follows:

$$\begin{aligned} S &= \left\lceil \frac{100}{3} \right\rceil \times \left\lceil \frac{50 \times 3}{128} \right\rceil \\ &= [34] \times [2] \\ &= 68 \text{ sectors} \end{aligned}$$



FILE/DEVICE RELATIONSHIPS

Devices required by files are allocated automatically by MPE/3000. The user can specify these devices by type (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 the user's program asks to read that file, it references the file by its formal file designator. MPE/3000 then determines the device on which the file resides, or its disc address if applicable, and accesses it for the user. When the user's program writes information to a particular file to be output on a device such as a line printer, again the program refers to the file by its formal file designator; MPE/3000 then automatically allocates the required device to that file. Throughout its existence, every file remains device-independent; that is, it is always referenced by the same formal file designator regardless of where it currently resides. The user's program always deals with logical records.

NON-SHARABLE DEVICE ACCESS

The specification of a device by type when a file is opened implies a request for the initial allocation of a previously unopened device. (The device specification is ignored when \$STDIN[X] and \$STDLIST are opened.) A job may *reallocate* an opened device by specifying the device's logical device number when its file is opened.

Multiple processes can asynchronously interleave accesses to reallocated devices. Since the file system does "anticipatory reads" on buffered input devices, multiple processes should specify Inhibit Buffering if records must be transmitted in the same order as requested.

FILE DOMAINS

The set of all permanent disc files in MPE/3000 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 the user with an account and group which provides the basis for his local file references. The user may be required to supply *passwords* for the account and group to log-on, but thereafter (if the normal (default) MPE/3000 file security provisions are in effect) he can:

- Have unlimited access to any file within his log-on or home group. (If, however, the file is protected by a *lockword*, the user must know this lockword.)

- Read, and execute programs residing in, any file in the public group of his account, and in the public group of the system account.

As noted later, the default MPE/3000 file security provisions can be overridden at the account, group, and/or file level to provide greater or lesser file access restrictions.

Potentially, if the MPE/3000 file security provisions at the account, group, and file levels were all suspended, and the user knew all account and group names and file lockwords, he could access any permanent file in the system once he logged-on. (Notice that once a user logs-on to an account and is associated with a group, he does *not* need to know the *passwords* for other accounts and groups to access files assigned to them — he only requires their account and group names. But, if any of these files are protected by a file lockword, the user must know that lockword.)

For every job or session running in the system, MPE/3000 recognizes another file domain, called the *job or session file domain*. This domain contains all temporary files opened and closed within the job or session without being saved (declared permanent). Files in these domains are deleted when the job or session terminates (if they are job/session temporary files), or when the creating program ends (if they are regular temporary files).

FILE LABELS

MPE/3000 reads and writes file labels for files on disc during allocation of the devices on which the files reside. The format and content of file labels is presented in Appendix F.

FILE ACCESSING

The user accesses files through commands and intrinsic calls. Commands, described in this section, are issued external to the user's program and perform general functions, such as creating, deleting, or listing a file. Intrinsic calls, described in the next section, are issued *programmatically* (within a user's program). Generally, files are opened (through the FOPEN intrinsic); operated on through various intrinsics that read information from them, write information to them, update them, or manipulate them; and finally, they are closed (through the FCLOSE intrinsic).

Within a user program, an MPE/3000 system program such as a compiler, or a command executor, a file is accessed by its *formal file designator* — the name by which the particular program recognizes the file. (In SPL/3000, this is the name associated with it by the FOPEN intrinsic.) At program execution time, this formal file designator is always associated or equated with an *actual file designator* — the name of the actual file to be used and the physical device upon which it resides, as recognized throughout the system by MPE/3000. Thus, the actual file designator is an execute-time re-definition of the file specified in the program by the formal file designator. If the user does not specify an actual file designator for a formal file designator, MPE/3000 uses the formal file designator for the actual file designator.

MPE/3000 recognizes actual file designators for four types of files:

- System-Defined Files
- User Pre-Defined Files

- New Files
- Old Files

The programmer can specify any of these designators programmatically.

System-Defined Files

System-defined file designators indicate those files that MPE/3000 uniquely identifies as standard input/output devices for a job/session. They are referenced as follows:

| Actual File Designator | Device/File Referenced |
|------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| \$STDIN | A file name indicating the standard job or session input device (that from which the job or session is initiated). For a job, this is typically a card reader. For a session, this is typically a terminal. Input data images in the \$STDIN file should not contain a colon in column 1, since this indicates the end-of-data. (When data is to be delimited, this should be done through the :EOD command, which performs no other function.) |
| \$STDINX | Equivalent to \$STDIN, except that MPE/3000 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 :JOB, :DATA, :EOJ, and :EOD are exceptions that always indicate the end-of-data but are otherwise ignored in this context; they are <i>never</i> read as data.) |
| \$STDLIST | A file name indicating the standard job or session listing device (customarily a printer for a batch job and a terminal for a session). |
| \$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/3000 but no physical output is actually performed. Thus, \$NULL can be used to discard unneeded output from a running program. |

User Pre-Defined Files

A user pre-defined file is any file that was previously defined or re-defined in a :FILE command, as discussed later in this section. In other words, it is a back-reference to that :FILE command. It is referenced by the following file designator format:

**formaldesignator*

formaldesignator The name used in the *formaldesignator* parameter of the :FILE command.

New Files

New files are files that have not yet been created, and are being created/opened for the first time by the current program. New files can have the following actual file designators:

| Actual File Designator | File Referenced |
|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| \$NEWPASS | A temporary disc file that can be automatically passed to any succeeding MPE/3000 command within the same job/session, which references it by the file name \$OLDPASS. Only one such file with this designation can exist in the job/session at any one time. (When \$NEWPASS is closed, its name is automatically changed to \$OLDPASS, and any previous file named \$OLDPASS in the job/session is deleted. (<i>Passing</i> is explained in later examples.)) |
| <i>filereference</i> | Unless the user specifies otherwise, this is a temporary file, residing on disc, that is destroyed on termination of the creating program. If closed as a job/session temporary file, as shown later in this section, it is purged at the end of the job/session. If closed as a permanent file, it is saved until purged by the user. Typically, this format consists of a file name containing up to eight alphanumeric characters, beginning with a letter, as discussed below. In addition, other elements (such as a group name, account name or lockword) can be specified. |

Old Files

Old files are existing named files presently in the system. They may be named by the following designators:

| Actual File Designator | File Referenced |
|------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| \$OLDPASS | The name of the temporary file last closed as \$NEWPASS. |
| <i>filereference</i> | Any other old file to which the user has access. (The <i>filereference</i> format is discussed below.) It may be a job/session temporary file created in this or a previous program in the current job/session, a permanent file saved by any program, or a permanent file built (with the :BUILD command) in any job/session. |

Filereference Formats

When the user references a file by the *filereference* designator, he writes *filereference* in any of three formats. (In no case, however, can *filereference* exceed a total of 35 characters.)

The format most commonly used applies to files contained in the user's log-on group:

filename [/lockword]

In this format, *filename* is the name of the file. It is written as a string of up to eight alphanumeric characters, beginning with a letter. (Because a file reference is always qualified by the group and account to which the file belongs, individual file names must be unique only within the file's group.) The lockword need only be specified when referencing an existing (old) disc file protected by a lockword assigned by the user who created the file. Neither the *filename* nor *lockword* should contain embedded blanks. Additionally, the slash mark (/) that separates these two elements should not be preceded nor followed by blanks.

NOTE: Whenever a file is referenced in any command, or in the FOPEN intrinsic that opens the file (described in Section VI), the lockword (if any) must always be supplied — even if the accessor is the creator of the file.

EXAMPLE:

The following three examples illustrate this filereference format:

OUTPUT

AL126797

PAYROLL/X229AD

The format used when the programmer references a file in his log-on account but within another group is:

filename [/lockword].groupname

Embedded blanks are not permitted. The group to which the file belongs is designated by *groupname*. As an example, if a user logs-on under a group other than his home group, but wants to reference a file in the home group, he must enter the name of the home group as the *groupname* specifier.

Remember that the file *lockword* relates only to the ability to access files, and not to the account and group *passwords* used during log-on.

EXAMPLES:

These file references include groupname specifiers:

```
FILEB.GP2
X3.PUR
GOFILE/Z22.GR07
```

The third *filereference* format is

filename [/lockword].groupname.accountname

Embedded blanks are not permitted. This format is used to reference a file assigned to a group that belongs to an account that is *not* the user's log-on account. The account is specified by *accountname*.

EXAMPLE:

The following file references contain accountname specifiers:

```
SMITH.GRP7.ACCT47
FILLER.PUB.SYS
NB3/X23DG.GRP5.ACCT190
```

Lockwords

As noted earlier, the creator of a disc file can assign to it a lockword which must thereafter be supplied to access the file in any way. This lockword is independent of, and serves in addition to, the other basic security provisions governing the file. The lockword is assigned by including it in the *filereference* parameter used when the file is created. It can be subsequently changed by the :RENAME command or the FRENAME intrinsic, discussed later. (:RENAME and FRENAME are also used to initially assign a lockword to an existing file.) At any time, a file can have only one lockword. Only the creator of a file can change a lockword for that file.

EXAMPLE:

To assign the lockword SESAME to a new file named FILEA, the user enters 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
```

When initially requesting access to an old disc file protected by a lockword, the user must supply the lockword in the following manner:

- In batch mode, as part of a file name specified in the :FILE command or FOPEN intrinsic call that establishes file access. If a file is protected by a lockword, but no lockword is supplied, access is not granted.
- In session mode, as part of a file name specified in the :FILE command, or FOPEN intrinsic call. If a file is protected by a lockword but no lockword is supplied when the file is opened, MPE/3000 interactively requests the user to supply the lockword, as follows:

LOCKWORD: filename.groupname.accountname?

EXAMPLE:

In the following :FILE command, the old file XREF (protected by lockword OKAY) is referenced.

```
:FILE INPUT=XREF/OKAY, OLD
```

Remember that the lockword is always separated from the filename by a slash.

On terminals with the ESC (Escape) key facility, the user can inhibit the echo facility (and thus suppress printing of the lockword) prior to entering the lockword by pressing the ESC and ; keys.

After he enters the lockword, he can restore echoing by pressing the ESC and : keys.

Where the terminal does not support dynamic half-duplex operation, the user can conceal a lockword simply by omitting it from the file reference. In this case, MPE/3000 outputs a request for the lockword, directs the carriage to skip a line, prints an eight-character mask, and returns the carriage to the beginning of the mask. The user then enters the lockword, which is echoed on top of the mask to ensure privacy.

File System Accounting

When the MPE/3000 default accounting provisions are in effect at the account and group level, the amount of permanent disc file space accumulated by users is monitored by MPE/3000 but is not limited. (The default provisions are described fully in *MPE/3000 Operating System, System Manager/System Supervisor Manual*. However, limits on the amount of permanent file space allotted can be established at the account level (by System Managers Users) and the group level (by Account Manager Users). The limits are established in terms of disc sectors. When an attempt is made to save a new disc file or to create, rename, or add extents to a permanent file, if either the account or group disc file space count exceeds the current limit, the file request is denied. Otherwise, the account and group disc file space counts are updated.

Input/Output Sets

All file designators described above can be classified as those used for input files (*Input Set*) and those used for output files (*Output Set*). These sets, referred to frequently throughout this manual, are defined as follows:

Input Set

| | |
|--------------------------|-----------------------------------------------------------------------------------------|
| \$STDIN | The job/session input device. |
| \$STDINX | The job/session input device with commands allowed. |
| \$OLDPASS | The last \$NEWPASS file closed. |
| \$NULL | A constantly-empty file that will return an end-of-file indication whenever it is read. |
| <i>*formaldesignator</i> | A back-reference to a previously-defined file. |
| <i>filereference</i> | A file name, and perhaps account and group names and a lockword. |

Output Set

| | |
|--------------------------|-----------------------------------------------------------------------------------------------------|
| \$STDLIST | The job/session listing device. |
| \$OLDPASS | The last file passed. |
| \$NEWPASS | A new temporary file to be passed. |
| \$NULL | A constantly-empty file that returns a successful indication whenever information is written to it. |
| <i>*formaldesignator</i> | A back-reference to a previously-defined file. |
| <i>filereference</i> | A file name, and perhaps account and group names and a lockword. |

SPECIFYING FILE CHARACTERISTICS

Formal (programmatic) file designators can be equated with actual file designators through the :FILE command. This command enables the user to

- Write programs that reference files whose actual names and characteristics he may not yet know. This allows him to remain uncommitted to specific disc files or devices until run-time, when their names are equated with the user's programmatic references.

- Issue detailed file specifications at run-time that override any corresponding specifications declared within the user's program (through an FOPEN intrinsic call) or in a previous MPE/3000 :FILE command.

The specifications in a :FILE command do not take effect until the user's program is running and opens the file referenced. The :FILE command specifications hold throughout the entire job/session, unless superceded (by another :FILE command) or revoked by the user (through the :RESET command). At job or session termination, however, all :FILE commands are cancelled.

If the user does not include in the job stream any file command referencing a particular formal file designator named in his program, that formal file designator will be used as the actual file designator, and any file characteristics specified (explicitly or by default) within the program will apply.

When two (or more) :FILE commands referencing the same formal designator appear in a job/session, the second command replaces the first one.

The :FILE command can be written in any of the following formats, depending on the type of file referenced, and applies to files on disc, tape, or any other device.

For new files:

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

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

$\left[;CCTL \right]$
 $\left[;NOCCTL \right]$

$\left[;ACC = accesstype \right]$

$\left[;NOBUF \right]$
 $\left[;BUF [= numbuffers] \right]$

$\left[;EXC \right]$
 $\left[;EAR \right]$
 $\left[;SHR \right]$

$\left[;MR \right]$
 $\left[;NOMR \right]$

Command Parameters

The variable parameters for the :FILE command are

| | |
|-------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>formaldesignator</i> | The programmatic formal file designator, as referenced in the user's program. (Required parameter.) |
| <i>filereference</i> | A file name (and perhaps account and group names and a lockword) in the <i>filereference</i> format. If a new file is referenced, but the <i>filereference</i> parameter is omitted from the :FILE command, the name is equated to the formal designator. (Optional parameter.) |
| <i>NEW</i> | A specification that the file is a new file. (Required parameter if <i>filereference</i> is used in the <i>new file</i> format.) |
| <i>OLD</i> | A previously-existing permanent file saved in the system file domain. The file continues to exist after the current job/session terminates. (Optional parameter.) |
| <i>OLDTEMP</i> | A previously-existing temporary file in the job/session file domain. This file is deleted at the end of the current job/session. (Optional parameter.) |
| <i>resize</i> | The size of the logical records in the file. If a positive number, this represents <i>words</i> . If a negative number, this represents <i>bytes</i> . (If the records are undefined-length, this represents their maximum size. For variable-length records, the maximum size is <i>resize</i> x <i>blockfactor</i> .) The default value is a value supplied at configuration time. |

The values generally specified by HP are

| | | |
|-------------|---|------|
| Disc | = | 128 |
| Tape | = | 128 |
| Printer | = | -132 |
| Card Reader | = | -80 |
| Card Punch | = | -80 |
| Terminal | = | -72 |

(Optional parameter.)

| | |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>blockfactor</i> | The size of each buffer to be established for the file, specified as an integer equal to the number of logical records per block. (For fixed-length records, <i>blockfactor</i> is the actual number of records in a block. For variable-length records, <i>blockfactor</i> is interpreted as a multiplier used to compute the block size (maximum <i>resize</i> x <i>blockfactor</i>). For undefined-length records, <i>blockfactor</i> is always one logical record per block; any unused portion of the block is filled with ASCII blanks or binary zeros.) The <i>blockfactor</i> value specified may be overridden by MPE/3000. The default value is calculated by dividing the specified <i>resize</i> into the configured physical record size; this value is rounded downward to an integer that is never less than 1. (Optional parameter.) |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

| | |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>F</i> | Specifies fixed-length records. <i>F</i> is the default if <i>F</i> , <i>U</i> , or <i>V</i> is not specified. (Optional parameter.) |
| <i>U</i> | Specifies records of undefined length (no blocking). (Optional parameter.) |
| <i>V</i> | Specifies variable-length records. (No blocking on unit-record devices.) (Optional parameter.) |
| <i>BINARY</i> | Specifies binary-coded records. (Optional parameter.) |
| <i>ASCII</i> | Specifies ASCII-coded records. (Optional parameter.) |
| <i>CCTL</i> | Indicates that the user is supplying carriage-control characters with his write requests, valid for any ASCII list file. (Optional parameter.) |
| <i>NOCCTL</i> | Indicates that no carriage-control characters are supplied with write requests. (Optional parameter.) |
| <i>accesstype</i> | The type of access allowed users of this file: |
| <i>IN</i> | = Read-access only. (The <i>FWRITE</i> , <i>FUPDATE</i> and <i>FWRITEDIR</i> intrinsic calls cannot reference this file.) |
| <i>OUT</i> | = Write-access only. Any data on the file prior to the current <i>FOPEN</i> request is deleted. (The <i>FREAD</i> , <i>FREADSEEK</i> and <i>FREADDIR</i> intrinsic calls cannot reference this file.) |
| <i>OUTKEEP</i> | = Write-access only, but previous data in the file is not deleted. (The <i>FREAD</i> , <i>FREADSEEK</i> and <i>FREADDIR</i> intrinsics cannot reference this file.) |
| <i>APPEND</i> | = Append-access only. The <i>FREAD</i> , <i>FREADSEEK</i> , <i>FREADDIR</i> , <i>FPOINT</i> , <i>FSPACE</i> , and <i>FWRITEDIR</i> intrinsic calls cannot be issued for this file. |
| <i>INOUT</i> | = Input/output access. Any file intrinsic except <i>FUPDATE</i> can be issued against this file. |
| <i>UPDATE</i> | = Update access. All file intrinsics, including <i>FUPDATE</i> , can be issued for this file. |

If *accesstype* is omitted in the `:FILE` command (and in the FOPEN intrinsic call that opens the file), the default value assigned is IN for all devices except output-only devices such as card punches, printers paper tape punches, and plotters.

If a process attempts to violate the *accesstype* of a file, an error is returned. (Optional parameter.)

NOBUF

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

numbuffers

The number of buffers to be allocated to the file. This must be an integer value. The maximum value is 16. If omitted or set to 0, a default value of 2 is assigned. This parameter is not used for files representing interactive terminals, since a system-managed buffering method is always used in such cases. (Optional parameter.)

EXC

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

EAR

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

SHR

After the file is opened, permits concurrent access to this file through another FOPEN request issued by this or another process. (Optional parameter.)

MR

Specifies that individual read or write requests are not confined to record boundaries, as explained under the FOPEN request discussion in Section VI. (Restricted to NOBUF files.) (Optional parameter.)

NOMR

Specifies that individual read or write requests are confined to record boundaries. (Optional parameter.)

DEL

Specifies that the file is to have regular temporary file disposition; it will be deleted when the user's program closes it.

SAVE

Specifies that the file is to have permanent file disposition; when the user's program closes it, the file remains in the system file domain, potentially available to other users. (Optional parameter.)

TEMP

Specifies that the file is to have job/session temporary file disposition. When the user's program closes the file, it remains in the job/session file domain. But when the job/session terminates, the file is deleted. (Optional parameter.)

NOTE: If DEL, SAVE, or TEMP is not 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 is used to make a non-specific, generic reference to a *type* of device (such as any disc drive or magnetic tape unit). (These names are defined and related to specific sets of devices when the system is configured. All 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 default is DISC.

The logical device number refers to a specific 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, a user would specify the logical device number of a specific device when running a hardware diagnostic program for checking that device. Note that 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). (Optional parameter.)

outputpriority

(For spooled output devices.) The “output priority” to be attached to this file. This priority is used to determine the order in which files will be produced, when several are waiting for the same device. This must be a number between 1 (lowest priority) and 13 (highest priority), inclusive. If this value is less than the current output fence set by the console operator, the file will be deferred from printing/punching until the operator raises the *outputpriority* of the file or lowers the output fence. This parameter is ignored for non-spooled output devices. The default value is 8. (Optional parameter.)

numcopies

(For spooled output devices.) The number of copies of the entire file to be produced by the spooling facility. The copies will not appear contiguously if the console operator intervenes or if a file of higher *outputpriority* becomes READY before the last copy is complete. This parameter is ignored for non-spooled output devices. The default value is 1. (Optional parameter.)

filecode

An integer or mnemonic recorded in the file label and made available to processes accessing the file through the FGETINFO intrinsic (Section VI). If an integer is used, it must be a positive value ranging from 0 to 1023, or one of the HP-defined integers shown below. These HP-defined integers also have corresponding mnemonics that can be used in their place:

| <u>Mnemonic</u> | <u>HP-defined Integer</u> | <u>Defines the File as:</u> |
|-----------------|---------------------------|----------------------------------|
| USL | 1024 | A USL file. |
| BASD | 1025 | A BASIC/3000 data file. |
| BASP | 1026 | A BASIC/3000 program file. |
| BASFP | 1027 | A BASIC/3000 fast program file. |
| RL | 1028 | A relocatable library (RL) file. |
| PROG | 1029 | A program file. |
| STAR | 1030 | A STAR/3000 file. |
| SL | 1031 | A segmented library (SL) file. |

If this parameter is omitted from the :FILE command, and from the FOPEN intrinsic call that opens the file, the default value is 0. (Optional parameter.)

filesize

The total maximum file capacity, specified only for a NEW file, in terms of physical records (for files containing variable-length and undefined-length records), and logical records (for files containing fixed-length records). Must be a double-word integer. The default value is 1023. The maximum capacity allowed is 184,000 sectors. (The number of sectors in a file is found by the formula shown earlier under *FILE CHARACTERISTICS*.) (Optional parameter.)

numextents

The 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 *filesize* parameter value divided by the *numextents* parameter value. If specified, *numextents* must be an integer value from 1 to 16. The default is 8. (Optional parameter.)

initialloc

The number of extents to be allocated to the file *at the time it is opened*. This must be an integer from 1 to 16. The default value is 1. If an attempt to allocate the requested space fails, the FOPEN intrinsic that opens the file returns an error condition code to the user's program when that program runs. (Optional parameter.)

**formaldesignator₁*

The name of a file defined in a previous :FILE command, preceded by an asterisk.

Accessing Files Already in Use

When a user's process attempts to access a file already being accessed by another process, the action taken by MPE/3000 depends on the current use of the file, as shown in Figure 5-1.

REQUESTED ACCESS GRANTED, UNLESS NOTED

| Requested Access | Current Use | FOPENed for Input | | FOPENed for Output | | FOPENed for Input/Output | | Program File Loaded | Being :STOREd | Being :RESTOREd |
|------------------------|-------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|-----------------|
| | | SHR | EAR | SHR | EAR | SHR | EAR | | | |
| FOPEN for Input | SHR | 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 91 |
| | EAR | Requested Access Granted | Requested Access Granted | Error 90 | Error 90 | Error 90 | Error 90 | Error 90 | Error 90 | Error 91 |
| FOPEN for Output | SHR | Requested Access Granted | Error 91 | Requested Access Granted | Error 91 | Requested Access Granted | Error 91 | Error 91 | Error 91 | Error 91 |
| | EAR | Requested Access Granted | Error 91 | Error 90 | Error 90 | Error 90 | Error 90 | Error 90 | Error 90 | Error 91 |
| FOPEN for Input/Output | SHR | Requested Access Granted | Input Granted | Requested Access Granted | Input Granted | Requested Access Granted | Input Granted | Input Granted | Input Granted | Error 91 |
| | EAR | Requested Access Granted | Input Granted | Error 90 | Error 90 | Error 90 | Error 90 | Error 90 | Error 90 | Error 91 |
| :RUN,CREATE | | Requested Access Granted | | Error Message | | Error Message | | Requested Access Granted | Only if Loaded | Error Message |
| :STORE | | Requested Access Granted | | Error Message | | Error Message | | Requested Access Granted | Error Message | Error Message |
| :RESTORE | | Error Message | | Error Message | | Error Message | | Error Message | Error Message | Error Message |

- NOTES: 1. SHR = Share; EAR = Exclusive, allow reading.
 2. Fully exclusive accesses cause any succeeding access (except :STORE) to fail.
 3. Append access treated like output; Update treated like input/output.
 4. Error 90 = Calling process requested exclusive access to a file to which another process has access.
 5. Error 91 = Calling process requested access to a file to which another process has exclusive access.

Figure 5-1. Actions Resulting From Multi-access of Files

Re-Specifying File Names

One common application of the `:FILE` command is to allow the user to reference, within his program, files whose names and characteristics he may not know at the time he writes his program.

EXAMPLE:

Suppose, for example, that the user is writing a generalized program that reads input from a different file each time it is executed. Within the program, the user could reference the input file by the formal designator `INPUT`, and an output file by the formal designator `OUTPUT`. After the program is compiled and prepared on disc, the user could issue `:FILE` commands to equate `INPUT` and `OUTPUT` with the corresponding actual file designators of the files to be used on this run. Then, he could issue the command to execute the program. Suppose that the user wanted to execute such a program (called `PROG1`) with an actual file named `MYFILE` used as input. He also wants to designate a disc file actually named `TEMP` for output. To do this, he could enter the following:

```
•  
•  
•  
;FILE INPUT=MYFILE  
;FILE OUTPUT=TEMP  
;RUN PROG1  
•  
•  
•
```


Passing Files

Another example illustrates how the :FILE command can be used to pass files between programs.

EXAMPLE:

In this example, 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  
:RUN PROG2  
•  
•  
•
```

Issuing Detailed File Specifications

The user can use the :FILE command to designate, for a file, various detailed specifications (such as device type, file type, code, or access mode). This command overrides any existing specifications defined for the file in the user's program.

EXAMPLE:

Suppose a BASIC/3000 user is accessing MPE/3000 from a terminal. He knows that during the course of his session, he will want to list his program and its output on a line printer. Before calling the BASIC/3000 Interpreter, he uses the :FILE command to specify a file for printout as follows:

```
      .  
      .  
      .  
➔ :FILE PRINTER; DEV=LP  
  :BASIC ,,*PRINTER  
      .  
      .  
      .  
    >10 FOR I = 1 TO 10  
      .  
      .  
      .  
    >LIST, 10-50  
      .  
      .  
      .
```

The :FILE command will direct the BASIC/3000 program listing and output to a line printer file. (Once the user has entered BASIC, he may transmit output from his program to the line printer by entering the LIST command.)

All :FILE commands must precede the :RUN or subsystem execution commands to which they apply. Each :FILE command remains active throughout the entire job or session unless that command is revoked or superceded. For this reason, the same program cannot be executed twice within the same job if that program references different files each time — unless a new :FILE command or a :RESET command is issued prior to the second execution.

As another example of this technique:

EXAMPLE:

Two FORTRAN/3000 programs are to be executed within one job. Before the first program is executed, a new file must be created with the following specifications: a binary disc file, direct access, with fixed-length records of 100 words each. Maximum size is 5000 records, divided into 10 extents, but only one extent (500 records) is to be allocated initially. The initial FORTRAN/3000 program expects this file to be accessed as logical unit number 8. The user wishes to save the file under the name FDATA1. To specify this file, the user enters:

```
:FILE FTN08=FDATA1, NEW; SAVE; REC=100,, F, BINARY; DISC=5000, 10, 1
```

This file will be created when opened by the :FORTRAN program; it will be permanently saved under the user's log-on group when closed by the FORTRAN program.

The first FORTRAN program may now be executed, accessing this file for disc input/output. Before the second program is executed, however, the :FILE command must be used again since the second program expects to access this file through logical unit numbers 20 and 21. Rather than re-enter the above specification, which is quite lengthy, the user simply enters these references and then executes his program:

```
:FILE FTN20=FDATA1  
:FILE FTN21=*FTN20      (OR, :FILE FTN21=FDATA1)
```

Implicit :FILE Commands

As mentioned earlier, compilers and other subsystems that run under MPE/3000 accept actual file designators as parameters in the commands that call these programs. Although the user is 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/3000 compiler, the formal file designator for the *textfile* input is FTNTEXT. In the :FORTRAN command, this is related to the *textfile* parameter shown below:

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

When the user specifies:

```
:FORTRAN ALSFILE
```

MPE/3000 implicitly issues the following :FILE command, invisible to the user:

```
:FILE FTNTEXT=ALSFILE
```

When calling a compiler or subsystem, any valid actual file designators (except the formal designator used by the subsystem) can be used as command parameters, including those of the **formaldesignator* (back-reference) format.

EXAMPLE:

In the following code, the user specifies 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 the user's previous :FILE command:

```
:FILE FTNTEXT=*SOURCE
```

When a compiler or subsystem terminates, MPE/3000 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 designator and the user's.

Command File Parameters

The programmer can avoid issuing :FILE commands that conflict with certain pre-defined :FILE commands, if he knows the formal file designators used by MPE/3000 subsystems and commands. The formal file designators for compilers, interpreter, and other subsystem commands are:

| 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 |

| 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 |
| :STAR | <i>listfile</i> | STRLIST |

| Command | Parameters | Formal File Designator |
|---------|-------------------------------------------------------------------------------|----------------------------------------|
| :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 are as follows. Implicit :FILE commands are not issued for these designators. Rather, the formal designator specifies the destination of listings generated by the command executor if the user does not re-specify the destination by issuing an *explicit* file command. (This :FILE command remains in effect throughout the job/session, unless an applicable :RESET command or another :FILE command is issued.)

| Command | Parameters | Formal File Designator |
|--------------------|------------|------------------------|
| :PREP :PREPRUN | PMAP | SEGLIST |
| :PREPRUN :RUN | | |
| :RESTORE :STORE | SHOW | SYSLIST |

RESETTING A FORMAL FILE DESIGNATOR

A formal file designator referenced in a prior `:FILE` command can be reset to the meaning defined by its original actual file designator. This is requested by issuing the `:RESET` command. The `:RESET` command effectively nullifies any previous explicit or implicit `:FILE` command referencing that formal designator, and applies to files on disc, tape, or any other device. The format of the `:RESET` command is

```
:RESET {formaldesignator}
      @
```

`formaldesignator` The formal file designator to be reset.

`@` An indication that the formal file designators referenced in all prior `:FILE` commands in the job/session are to be reset.

(Either the `formaldesignator` or `@` parameter must be entered.)

EXAMPLE:

Suppose that a user runs two programs, both referencing a file defined within these programs by the designator `DFILE`, a new, temporary file on disc. Before he runs the first program, the user wants to redefine the file so that it is output to the standard list device. To do this, he issues a `:FILE` command equating `DFILE` with `$STDLIST`. In the second program, the file is again to be a temporary file on disc. The user issues a `:RESET` command to re-establish the programmatic specifications:

```
      :JOB JNAME, UNAME.ANAME
      .
      .
      .
      ▶ :FILE DFILE=$STDLIST
      ▶ :RUN PROG1
      :RESET DFILE
      :RUN PROG2
      .
      .
      .
      :EOJ
```


CREATING A NEW FILE

The programmer can create a new disc file and specify its characteristics by issuing the :BUILD command. This command, when encountered, results in the *immediate* allocation of an empty file having the specifications supplied by the user as command parameters. (This is unlike the :FILE command, which does not take effect until the file is opened by the user's program and whose specifications override those supplied by the FOPEN intrinsic.) The user can specify the new file as a job/session temporary file, or as a permanent file.

To issue the :BUILD command, the user must have SAVE access (as defined later in this section) to the group to which the new file is to belong (unless the TEMP parameter is specified). The :BUILD command is written in this format:

:BUILD filereference

$$\left[;REC = \left[\textit{resize} \right] \left[, \left[\textit{blockfactor} \right] \left[, \left[\begin{array}{c} F \\ U \\ V \end{array} \right] \left[\begin{array}{c} ,BINARY \\ ,ASCII \end{array} \right] \right] \right] \right]$$

[;CCTL]

[;NOCCTL]

[;TEMP]

[;DEV = device]

[;CODE = filecode]

[;DISC = [filesize] [, [numextents] [, initalloc]]]

filereference

The filename in the *filereference* format, optionally including the account and group identifiers, and lockword. If specified, the account name must be that of the log-on account. (Required parameter.)

resize
blockfactor
F
U
V
BINARY
ASCII
CCTL
NOCCTL
device
filesize
numextents
initalloc
filecode

The same meanings and default values as the corresponding parameters described in the :FILE command discussion. (Optional parameters.)

TEMP

A specification that the file is a job *temporary* file, entered in the job/session temporary file directory; when job/session terminates, the file is deleted. If TEMP is omitted, the file is declared *permanent*; it is saved in the system file domain. (Optional Parameter.)

EXAMPLE:

To create a new job/session temporary file named NFILE, containing 500 records of 160 words each, the user enters:

```
!BUILD NFILE; DISC=500; REC=160; TEMP
```

SAVING A FILE

A temporary disc file can be changed to a permanent file by issuing the :SAVE command, immediately following execution of the program that opens the file. (To use this command, the user must have SAVE access to the group to which the referenced file is to belong.)

$$:\text{SAVE} \left\{ \begin{array}{l} \$\text{OLDPASS}, \text{newfilereference} \\ \text{tempfilereference} \end{array} \right\}$$

\$\$OLDPASS

The file currently being passed, used only if this is the file to be made permanent. (After it is saved, no file is in the pass condition.) (Optional parameter.)

newfilereference

The new file name to be assigned to \$\$OLDPASS when it is made permanent. (Required if \$\$OLDPASS is used.)

tempfilereference

The name of the temporary file to be made permanent under the same filename. This file is deleted from the temporary file domain. (Optional parameter.)

If no account and group identifiers are entered as part of the *newfilereference* or *tempfilereference* parameters, the file will be assigned to the log-on account and group. (If the account name is specified, it must be the name of the log-on account.)

EXAMPLES:

The following command assigns the most recently \$PASSED file to permanent status under the name PERMFILE: (Now, \$\$OLDPASS can no longer be referenced.)

```
!SAVE $OLDPASS, PERMFILE
```

The next command changes the temporary file DATAFILE.GROUPX to permanent status.

```
!SAVE DATAFILE.GROUPX
```

DELETING A FILE

To delete a disc file from the system, the user enters the :PURGE command:

:PURGE filereference[,TEMP]

filereference The name (and, if required, the group, account and lockword) of the file to be deleted. This must be a file to which the user has write (W) access, as defined in the MPE/3000 security provisions. (Required parameter.)

TEMP An indication that the file is a temporary job file. (Required parameter for temporary files.)

EXAMPLE:

The following command deletes the temporary file TFILE:

:PURGE TFILE, TEMP

LISTING FILE SETS

The user can obtain descriptions of one or more disc files in the system by issuing the :LISTF command. This command provides a listing that shows, for each file referenced, various items requested at the user's option. Among these items are: the file name, the type, size, number of records it contains, and other information found in the file label. To obtain this information for a file, the user need not have access to the file. Therefore, when a user issues a command to list descriptions of several files, the resulting listing may contain information about some files that are not accessible to him.

The :LISTF command is issued in this format:

:LISTF [fileset] [,detail] [;listfile]

fileset One or more files, referenced by filename, group, and account, as described below. If this parameter is omitted, all files in the user's log-on group are listed. (Optional parameter.)

detail A number indicating the amount and type of information to be listed, as follows:

- 0 = The filename. (An asterisk (*) denotes that the file is open.)
- 1 = The above, plus the file code, record size (bytes), record type (F, U, or V), whether ASCII or binary records (A or B), whether

carriage control option is taken (C, if so), the current end-of-file pointer, and the maximum number of records allowed in the file.

- 2 = The above, plus the blocking factor, number of disc sectors in use (including the file label and user headers), the number of extents currently allocated, the maximum number of extents allowed.
- 1 = An octal listing of the file label (if the user has System Manager or Account Manager Capability). Account-Manager Users can issue only :LISTF, -1 for files in their account; they cannot specify an account in *fileset* for this detail. (The format of file labels is shown in Appendix E.)

A *detail* specification greater than 2 defaults to 2; a *detail* specification less than -1 defaults to -1.

The default parameter is 0. (Optional parameter.)

listfile

The file on which the listing is to be written. This must be an ASCII file from the output set. It is automatically specified as a new, ASCII file with variable-length records, user-supplied carriage control characters (CCTL), OUT access type, and the EXC option; all other specifications are the same as the :FILE command default specifications. If omitted, \$STDLIST is assigned by default. (Optional parameter.)

The *fileset* parameter allows the user to request descriptions of one file alone, or various sets of files. It contains three positional-fields separated by periods: the file, group, and account fields. The file field permits the user to indicate a specific file or all files within the units designated by the other fields. The group field denotes the group to which the files belong. This can be the user's log-on group, any other group, or all groups within the accounts specified by the account field. The account field indicates the account or accounts to which the groups belong. This can be the log-on account, any other account, or all accounts in the system. (To specify *all* files, groups, or accounts, the user enters the character @ in the appropriate field. The omission of an entry in the group or account field denotes the log-on group or account.) For the *fileset* parameter, the following combinations of entries are possible.

| File Field | Group Field | Account Field | Entry Example | Meaning |
|-----------------|------------------|--------------------|-----------------|-------------------------------------------------------------------|
| <i>filename</i> | <i>groupname</i> | <i>accountname</i> | FILE.GROUP.ACCT | The file named, in the group and account designated. |
| <i>filename</i> | <i>groupname</i> | | FILE.GROUP | The file named, in the group designated under the log-on account. |
| <i>filename</i> | | | FILE | The file named, under the log-on group. |
| @ | <i>groupname</i> | <i>accountname</i> | @.GROUP.ACCT | All files in the group named, under the designated account. |
| @ | <i>groupname</i> | | @.GROUP | All files in the group named, under the log-on account. |
| @ | | | @ | All files in the log-on group. |
| @ | @ | <i>accountname</i> | @.@.ACCT | All files in all groups under the account named. |
| @ | @ | | @.@ | All files in all groups under the log-on account. |
| @ | @ | @ | @.@.@ | All files in the system. |

The following example shows how the complete :LISTF command is used:

EXAMPLE:

To list the file name, file code, size, type of records, ASCII vs. binary code, carriage-control option, end-of-file pointer, and maximum number of records for the file named BASE in the group USERGP under the programmer's log-on account, the following command is issued:

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

To list the filename of all files in all groups the user's log-on account, this command is entered.

```
:LISTF @.*@
```

DUMPING FILES OFF-LINE

To obtain a back-up copy of a particular user disc file or fileset, the user can copy the fileset off-line to a magnetic tape unit by issuing the :STORE command. 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).

Multi-file reels and multi-reel files are both supported by :STORE and :RESTORE.

NOTE: The :STORE and :RESTORE commands are used primarily as a back-up for files. They can be used 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 no destination device is specified in the :RESTORE command, MPE/3000 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.

The :STORE command is written as follows:

```
:STORE [filesetlist] ;tapefile[;SHOW][;FILES=maxfiles]
```

filesetlist A list of one or more files or filesets to be copied, written in this format:

```
[fileset[,fileset] ...]
```

In this list, *fileset* must be written as one of the combinations noted for *fileset* in the discussion of the `:LISTF` command. If the entire *filesetlist* parameter is omitted, the default value is `@` (all files of the log-on group are copied). If any file requires a lockword, the user may supply that lockword. If he fails to supply it (while in batch mode), the file will not be stored. If he fails to supply it (in interactive mode), he is prompted for the lockword. (Optional parameter.)

- tapefile* The name of the 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 **formaldesignator* format described earlier. (Required parameter.)
- SHOW* A request that the names of the stored files, plus the total number of files stored, the names of the files not stored, and the number of files not stored, be listed. If *SHOW* is omitted, only the total number of files stored, the names of the files not stored, and the number of files not stored, are listed. (Optional parameter.)
- maxfiles* The maximum number of files that may be stored. If omitted, the number 2000 is used by default. (Optional parameter.)

NOTE: Before issuing a `:STORE` command, the user must identify *tapefile* as a magnetic tape file. He does this through the `:FILE` command. In this case, the `:FILE` command should be written in the following format, including no parameters other than those shown:

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

The device parameter must be the device class name or logical unit number of a magnetic tape unit.

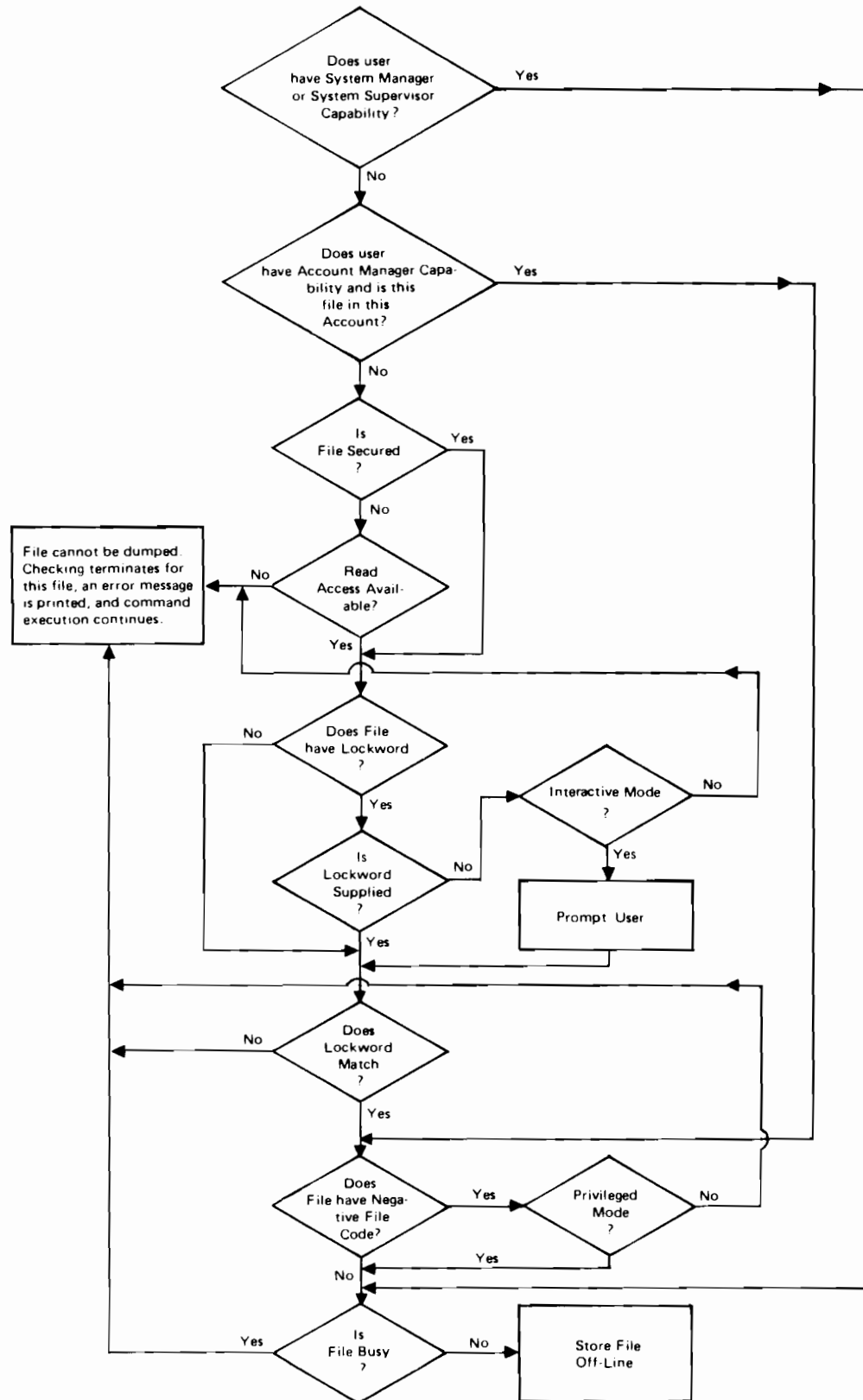
*All other parameters for *tapefile* are supplied by the `:STORE` command executor; if the user attempts to supply any of these himself, the `:STORE` command is rejected.*

A typical user can dump any file to which he has read-access. If a file has a negative file code, however, the user must have the *Privileged Mode Optional Capability*. A user with the System Manager or System Supervisor Capability can dump any user file in the system. An Account-Manager User can dump any file in his account (but cannot dump those with negative file codes unless he also has the privileged mode capability).

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 (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/3000 so that it cannot be altered or deleted until safely copied to tape. If the job performing the `:STORE` function is aborted (through the `=ABORTJOB` console command), some of the files being stored may remain locked until the next cold-load.

The flow chart below shows the checks performed against a file to ensure its eligibility for dumping:



After the tape is written, data showing the results of the :STORE command is printed. By default, this output is sent to the standard list device (\$STDLIST). However, the user can override this default and transmit the output to another file by issuing a :FILE command equating SYSLIST (the formal file designator by which the :STORE command executor references this list file) to another file. For example, a user at a terminal might transmit this output to a line printer by entering

```
:FILE SYSLIST=MYFILE; DEV=LP
```

If the *SHOW* parameter is omitted 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 following format:

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>filenamen</i> | <i>.groupnamen</i> | <i>.acctnamen</i> | <i>ldnn</i> | <i>addrn</i> |

FILES NOT STORED = yyy

| <i>FILE</i> | <i>.GROUP</i> | <i>.ACCOUNT</i> | <i>FILESET</i> | <i>CONDITION</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>filenamen</i> | <i>.groupnamen</i> | <i>.acctnamen</i> | <i>fileset#</i> | <i>msg</i> |

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 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 in the *filesetlist* parameter). The notation *msg* is a message denoting the reason that the file was not dumped, as follows. (These errors do not abort the file storing operation, which continues.)

| 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 | A user without Privileged Mode Capability is attempting to STORE a file with a negative file code. |
| LOCKWORD WRONG | The file lockword either was not provided or was specified incorrectly. |
| READ ACCESS FAILURE | The user does not have read access to the specified file. |
| TAPE READ ERROR | A tape read error has occurred on a block other than that containing the file label. |

The following catastrophic errors abort the :STORE command:

Command system error.

Disc input/output error (in system).

File directory error.

File system error on the tape file, list file, or temporary disc files used by the :STORE command executor.

The following example illustrates the use of the :STORE command.

EXAMPLE:

To copy all files in the group GR4X (in the user's log-on account) to a tape file named BACKUP, the user enters the following commands. A listing of the files copied appears on the standard list device.

```

:FILE BACKUP; DEV=TAPE
:STORE @.GP4M; *BACKUP; SHOW

```

Explicit or implicit redundancies are permitted in *fileselist*, but once a file has been locked down, any subsequent reference to it in *fileselist* results in the message BUSY even though execution of the :STORE command continues.

EXAMPLE:

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>acctname</i> | <i>fileset#</i> | <i>msg</i> |
|-----------------|------------------|-----------------|-----------------|-------------|
| <i>FN</i> | <i>GN</i> | <i>AN</i> | <i>2</i> | <i>BUSY</i> |

This same file will also be noted under *FILES STORED*.

RETRIEVING DUMPED FILES

The user can read back into the system, onto disc, any file, fileset, or filesetlist that has been stored off-line (on tape) by *:STORE*. The files referenced are attached to the appropriate groups and accounts, with previous account and group names, and lockwords all re-instated. File retrieval is requested with the *:RESTORE* command. This command does not create any new accounts or groups. Any tape file to be restored will only be restored if the account name and group name exist on disc (in the system directory).

```
:RESTORE tapefile [;filesetlist] [;KEEP] [;DEV=device] [;SHOW] [;FILES=maxfiles] . . .
```

- tapefile* The name of the tape file on which the *filesetlist* to be retrieved now resides. This file must be referenced in the **formaldesignator* format described earlier. A message is output to the console operator telling him which tape to mount, and the device it should be mounted on. (Required parameter.)
- filesetlist* The file, fileset, or filesetlist to be restored from tape. Each file is restored as a permanent file in the system file domain. The parameter is written in the same format, and subject to the same constraints as the *filesetlist* parameter of the *:STORE* command. 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 the *filesetlist* parameter is omitted, the default value is *@@@*. (all files on the tape). (Optional parameter.)
- KEEP* A specification that if a file referenced in *filesetlist* 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. 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* The device class name or logical device number of the device on which all files are to be restored. If omitted, an attempt is made to replace the files on the same device type (and subtype) on which

they were originally stored in the system; if this cannot be done, an attempt is made to restore the files to the same device type (ignoring sub-type); if this fails, an attempt is made to restore it to the device class DISC; if this fails, the file is not restored. (Optional parameter.)

SHOW A request that the names of the restored files, the total number of files restored, the names of the files not restored, and the total number of files not restored be listed. If *SHOW* is omitted, only the total number of files restored, a list of the files not restored, and a count of the files not restored, are listed. (Optional parameter.)

maxfiles The maximum number of files that may be restored. If omitted, the number 2000 is used by default. (Optional parameter.)

A user with System Manager or System Supervisor Capability 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. A user with Account-Manager Capability can restore any file in his account (but cannot restore those with negative file codes unless he also has the privileged mode capability). Any other user can restore any tape file in his log-on account if he has save-access to the group to which the file belongs (but cannot restore those with negative file codes unless he also has the privileged mode capability). If the file on tape is protected by a lockword, the lockword must be supplied in the :RESTORE command. (Users logged-on interactively are prompted for omitted lockwords.)

If a copy of a file to be restored already exists on disc, the user must have write access to the disc file (since it will be purged by :RESTORE). If this disc copy has a negative file code, the user must have privileged mode capability to restore it.

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 performed by the :STORE command, to ensure a file's eligibility for retrieval. If the SHOW parameter is included in the :RESTORE command, a listing is produced showing the names of the files restored, a count of files restored, a list of files not restored, and a count of files not restored. Otherwise, a count of files restored, a list of files not restored, and a count of files not restored, are supplied. As with the listing produced by :STORE, the listing output by :RESTORE is transmitted to a file whose formal designator is SYSLIST; if the user does not specify otherwise, this file is equated, by default, to the standard list device (\$STDLIST). The listing appears in the format shown below.

FILES RESTORED = 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>.</i> | | | | |
| <i>filenamen</i> | <i>.groupnamen</i> | <i>.acctnamen</i> | <i>ldnn</i> | <i>addrn</i> |

FILES NOT RESTORED = yyy

| FILE | .GROUP | .ACCOUNT | FILESET | CONDITION |
|------------------|--------------------|-------------------|-----------------|------------|
| <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>filenamen</i> | <i>.groupnamen</i> | <i>.acctnamen</i> | <i>fileset#</i> | <i>msg</i> |

In this format, *xxx* denotes the total number of files restored; *yyy* denotes the number of files requested that were not restored. The notations *filename*, *groupname*, and *acctname* under the FILES RESTORED heading name the individual files restored, and their groups and accounts, respectively. The notation *ldn* indicates the logical device number (in decimal) of the device on which the file now resides, and *addr* is the absolute address (in octal) of the file label. The notations *filename*, *groupname*, and *acctname* under the FILES NOT RESTORED heading, indicate the individual files not restored, and their groups and accounts. The notation *fileset#* shows the number of the fileset to which the particular file belongs (relative to its position in the *filesetlist* parameter.) The notation *msg* is an error message denoting the reason that the file was not restored, as follows. (These errors do not abort the file-restoring operation.)

| Message | Meaning |
|------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ACCOUNT DIFFERENT FROM LOGON | The file's account name is different from the name of the user's 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. (Examples of such catastrophic errors are listed below.) |

| Message | Meaning |
|------------------------------------|----------------------------------------------------------------------------------------------------------------------------|
| 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 the user does 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. |
| 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 | The user does 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 the user does not have Privileged Mode Capability. |
| TAPE FILE LOCKWORD WRONG | The tape file has a lockword that was not supplied by the user, or was specified incorrectly. |
| WRITE ACCESS FAILURE | The user does not have write-access to the copy of the file on disc. |

The following catastrophic errors abort the :RESTORE command:

Command syntax error.

Disc input/output error (in system).

File directory error.

File system error on the tape file (TAPE), list file (SYSLIST), or any of the three temporary files (GOOD, ERROR, and CANDIDAT) used by the :RESTORE command executor.

Improper tape; the tape used for input was not written in :STORE/:RESTORE format.

No continuation reel; the computer operator could not find a continuation reel for a multi-reel tape set.

Device reference error; the specification for the *device* parameter is illegal, or the device requested is not available.

Too many *filesets* specified.

EXAMPLE:

To retrieve from the file named *BACKUP* all files formerly belonging to the user's log-on group, the user enters:

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

If a tape file satisfying the @ specification already exists in the system, it is not restored.

NOTE: Tapes created by the *:STORE* command and *:SYSDUMP* command (discussed in *MPE/3000 Operating System, System Manager/System Supervisor Manual*), are compatible. Thus, a tape dumped through *:SYSDUMP* can be used as input for the *:RESTORE* command.

RENAMING A FILE

The user can change the name (*filereference*-format identity) of any disc file which he has created. When he does this, he effectively removes the file with the old name from the system and creates another file with identical contents and a new name. This command can be used also to move a file from one group to another by specifying different group names in the first two parameters, or to change the lockword of a file. Renaming is accomplished with the *:RENAME* command:

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

oldfilereference The current name of the file including optional group and account identifiers, and lockwords. (If an account is specified, it must be the log-on account.) (Required parameter.)

newfilereference The new name to be assigned to the file, including optional group and account identifiers, and lockwords. If an account identifier is specified, it must be that of the log-on account.

If a group identifier is used, it must be one to which the user has save access, as defined under the discussion of file security. If the group and account identifiers are omitted, the log-on group and account are assumed. (Required parameter.)

TEMP Indicates that the old file was, and the new file will be, a temporary file local to the job or session. (Optional parameter.)

NOTE: The user can apply the *:RENAME* command only to files that he himself has created.

EXAMPLES:

To change the name of a temporary job file from *OLDFILE* to *NEWFILE*. *ORGB*, the user enters:

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

To change the lockword of the permanent file *XFILE* from *LKD* to *BARX*, the user enters:

```
:RENAME XFILE/LKD, XFILE/BARX
```

SPECIFYING FILE SECURITY

As previously noted, when a user logs onto the system (or submits a job to it through the computer operator), he is 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/3000 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. Specifically, it permits use of the FLOCK and FUNLOCK intrinsics, and the exclusive-access option of the FOPEN intrinsic, all described in the next section. |
| 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 group</i> permanent, and to rename such files. 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. |

The types of users recognized by the MPE/3000 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. |

| User Type | Mnemonic Code | Meaning |
|----------------------|---------------|-----------------------------------------------------------------------------------------------------------------------------|
| Group Librarian User | 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 User always has R, A, W, L, X access to any file in the system, and S access to any group in his account; an Account Manager User always has unlimited (R, A, W, L, X) access to any file in his account, and S access to any group in his 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 or A access to that file is required.

NOTE: In addition to the above restrictions, in force at the account, group, and file level, a file lockword can be specified for each file. Users must then specify the lockword as part of the file-name 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 *MPE/3000 Operating System, 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 System Manager User 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), through which the System Manager User 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 User 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-Managing User 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/3000 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 individual) 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. A user must pass tests at all three levels — account, group, and file security, in that order — to successfully access a file in the requested mode.

If no security provisions are explicitly specified by the user, the following provisions are assigned 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, the standard user (without any other user attributes) has:

- Unlimited access (in all modes) to all files in his log-on group and home group.
- Reading and executing access (only) to all files in the public group of his account and the public group of the System Account.

The user cannot access any other file in the system (in any mode).

A user can only create files within his own account.

The legitimacy of a request to access a file is determined by checking the access mode requested by the user against the final access mode authorized for users of this type by the file security provisions. For various intrinsics, the functions requested versus the access mode necessary for the functions to be honored, are shown below. (The intrinsics are described in detail in Section VI.)

| Intrinsic | Function Requested | Access Mode Required to Honor Function |
|-----------|--------------------------------------------|----------------------------------------|
| FOPEN | Reading | R |
| | Writing | W |
| | Appending (only) | A (or W) |
| | Input/output | R and W |
| | Updating | R and W |
| | Exclusive accessing | L (or W or A) |
| | Semi-exclusive accessing | L (or W or A) |
| FLOCK | File locking | L (or W or A) |
| FUNLOCK | File unlocking | L (or W or A) |
| FCLOSE | Saving | S (relative to desired group) |
| | Any access beyond the current end-of-file. | 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* parameter of 0 for normal MPE/3000 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 next.

Changing File-Level Provisions

To change the security provisions assigned to any individual disc file, the creating user can enter the :ALTSEC command. This command permanently deletes all previous provisions specified for this file, and replaces them with those defined as the command parameters. The security provided by any *lockword*, however, is not affected. The :ALTSEC command format is:

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

where:

filereference

The name of the file whose security provisions are to be altered. This is any legal filename (including account and group names, if required.) The lockword, if any, must also be specified. (Required parameter.)

modelist

The modes of access that are permitted to the users specified in the immediately following *userlist* parameter. 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
- A = Appending (implicitly specifies L, also)
- W = Writing (implicitly specifies A and L, also)
- X = Executing

If any mode is omitted from this list, this implies (at the file level) that no one is permitted access in this mode. If no mode at all is specified, however, this implies (at the file level) completely unrestricted access to the file. Of course, access can also be limited by provisions specified at the account and group levels. (Optional parameter.)

NOTE: Even though the creator of a file may be barred from accessing the file by the modelist:userlist restrictions, he can still issue an :ALTSEC command against that file, and thus change the security provisions to allow him access.

userlist

The types of users to whom the access modes defined by the immediately-preceding *modelist* parameter apply. If two or more user types are specified, they should 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 that more than one *modelist:userlist* parameter combination can be used, to permit extremely versatile file-security specifications.

If no *modelist:userlist* parameter combination is specified, the following default security provisions are assigned: (R,A,W L,X: ANY).

EXAMPLES:

The following command alters the security provisions for the file named FILEX. This command permits the ability to read, execute, and append information to the file only to the creating user and the log-on or home-group users. (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, the user would enter:

```
:ALTSEC FILEX
```

The following :ALTSEC command changes the security provisions of :FILEX so that any group user can execute the file, but only the group librarian can read and write on it.

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

Suspending Security Provisions

From time to time, the creating user may wish to temporarily suspend all account, group, and file-level security provisions governing a disc file, to allow it to be accessed in any fashion by any user. (Note that this temporary suspension does not require the user to have the System or Account Manager Capability.) This is done with the :RELEASE command. (File lockword protection, however, is *not* removed by this command.) The command format is:

```
:RELEASE filereference
```

where:

filereference The name of the file whose security provisions are to be suspended. (As part of *filereference*, the lockword, account, and group may be specified.) (Required parameter.)

EXAMPLE:

The following command releases the security provisions for the file FILEX in the user's log-on group.

```
:_RELEASE FILEX
```

Restoring Security Provisions

To restore the security provisions suspended by the :RELEASE command, the creating user enters the :SECURE command:

```
:_SECURE filereference
```

where:

filereference The name of the disc file whose security provisions are to be restored. (The filereference can include the lockword, account, and group, as needed.) (Required parameter.)

EXAMPLE:

To restore the security provisions of FILEX, the user enters:

```
:_SECURE FILEX
```

LOCKING FILES

MPE/3000 allows several users to concurrently access a file. But occasionally, a programmer may want to prevent others from accessing such a file while he is performing some critical operation (such as updating) upon that file. To satisfy this need, MPE/3000 permits a user to lock out other users while he is accessing a file, on a continuous or a dynamic basis, by specifying certain parameters in the :FILE command or FOPEN intrinsic call that initiates file access. (These parameters and their use are described in the discussion of the FOPEN intrinsic call, in the next section.) To use these parameters, the programmer must be allowed the locking access mode noted earlier in this section.

FILE MANAGEMENT COMMAND FILE-TYPE SUMMARY

The file commands described in this section and the types of files/devices to which they apply are summarized below:

| All Devices | Disc Only | Disc Input, Tape Output | Tape Input, Disc Output |
|-----------------|----------------------------------------------------------------------------------|----------------------------|----------------------------|
| :FILE :RESET | :BUILD :SAVE :PURGE :LISTF :RENAME :ALTSEC :RELEASE :SECURE | :STORE | :RESTORE |



SECTION VI

Accessing and Altering Files

Within a user's program, the accessing and modification of files is requested through intrinsic calls. Each file referenced is first opened through the FOPEN intrinsic call. Then, other operations such as reading, writing, updating, and spacing forward or backward can be performed upon the file with other intrinsic calls. Finally, the file is closed through the FCLOSE intrinsic call, issued by the user's process or by MPE/3000 when the user's process terminates.

If the user is programming in SPL/3000, he declares the intrinsics and writes the intrinsic calls as he does other statements within his program, as illustrated in the examples throughout this section. If he is programming in another language, such as FORTRAN/3000 or BASIC/3000, any intrinsics required are called automatically by the commands in that language, or are invoked through other provisions described in the manuals covering those languages.

In the FOPEN intrinsic call, the user references a particular file by its formal file designator, described in the preceding section. When the FOPEN intrinsic is executed, it returns to the user's process a *file number* by which the system uniquely identifies the file. The file number, rather than the file designator, is used by subsequent intrinsics in referencing the file. In an SPL/3000 program, the user obtains this number through the normal conventions of that language. One such convention employs an SPL/3000 assignment statement to store the file number into a location specified by an identifier (name) which can then be used as an intrinsic call parameter to reference the file. The format of the assignment statement is discussed in the manual covering SPL/3000. An example is shown below:

EXAMPLE:

Suppose that a user issues an FOPEN call for a file designated FILLER. He could then store the file number assigned to FILLER in a location denoted by an identifier called F1 by writing an assignment statement. In this statement, the identifier appears in the left part and the FOPEN call appears in the right part. (FILLER is the identifier of a byte-array containing "FILLER.")

F1:=FOPEN(FILLER);

In subsequent intrinsic calls, the user refers to the file as F1. For example:

FCLOSE(F1,0,0);

Each intrinsic is declared and called as described in Section III. In this manual, each intrinsic call is shown within the context of its complete declaration head format; the call is distinguished from the remainder of the format by a box. The notation **OPTION VARIABLE** in the intrinsic head format indicates that certain parameters are optional; the optional parameters are shown in bold-face type. The absence of this notation means that all parameters are required.

The condition codes returned to the user's program by the file system intrinsics have the following general meanings. The specific meanings, of course, depend on the intrinsic:

| Condition Code | Meaning |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CCE | The function requested by the intrinsic call was completed successfully. |
| CCG | While servicing the request, MPE/3000 encountered the end of the file. |
| CCL | MPE/3000 could not service the request because an error occurred; corrective action may, in some cases, be taken. (By issuing an FCHECK intrinsic call, the user can have a more detailed error description transmitted to his process.) If, however, the error resulted from invalid parameters supplied by the user in the intrinsic call, the error is fatal and the user's process is aborted (or a software error trap, if previously enabled by the user, is activated). |

When a file is accessed by a process running a program written in a language other than SPL/3000, the file is generally (but not always) referenced by a file name. All intrinsic calls needed for opening, accessing, and closing the file are generated automatically by the user's process, and the file name is equated with the file number used by the intrinsics to reference the file.

When a new file is opened but not yet closed, it is always part of the job/session domain. At this time, the designator (SPL/3000 programs) or file name (programs in other languages) assigned by the user need not be unique. But when the file is saved, or closed without being deleted, MPE/3000 determines whether another file with the same designator name exists. If a name conflict occurs, a CCL condition code is returned to the user's process, and the specific error is made available through the FCHECK intrinsic. When a program aborts, old files are returned to the domain in which they were found when opened; new files are deleted.

NOTE: All intrinsics discussed in this section, with the exception of FOPEN, FGETINFO, and FRENAME, can be called (in privileged mode) with the DB register pointing to a data segment other than the calling process' stack. All parameters referenced in any calls to these intrinsics are always accessed using the current DB-register setting.

OPENING FILES

Before a user's process can read, write on, or otherwise manipulate a file, the process must initiate access to that file by opening it with the FOPEN intrinsic call. (This call applies to files on all devices.) When the FOPEN intrinsic is executed, it returns to the user's process the file number used to identify the file in subsequent intrinsic calls.

If the file is opened successfully (and the CCE condition code results), the file number returned is a positive integer ranging from 1 to 255. (Theoretically, one process may open a maximum of 255 files.) If the file cannot be opened (resulting in the CCL condition code), the file number returned is zero.

If a process issues more than one FOPEN call for the same file before it is closed, this results in multiple, logically-separate accesses of that file, and MPE/3000 returns a unique file number for each such access. Also, MPE/3000 maintains a separate logical record pointer (indicating the next sequential record to be accessed) for each such access.

In opening a file, FOPEN establishes a communication link between the file and the user's program by

- Allocating to the user's program the device on which the file resides. If the file resides on magnetic tape, FOPEN determines whether it is present in the system. (If it is not, FOPEN requests the system operator to supply the tape. Cataloging of tapes, however, is not done.) Generally, disc files can be shared concurrently among jobs and sessions. But magnetic tape and unit-record devices are allocated exclusively to the requesting job or session. For example, different processes within the same job may open and have concurrent access to files on the same magnetic tape or unit-record device; but this device cannot be accessed by another job until all accessing processes in this job have issued corresponding close requests (FCLOSE calls).
- Verifying the user's right to access the file under the security provisions existing at the account, group, and file levels.
- Determining that the file has not been allocated exclusively to another process (by the *exclusive* option in an FOPEN call issued by that process).
- Processing file labels (for files on disc). For new files on disc, FOPEN specifies the number of labels to be written.
- Allocating to the file the number of extents initially requested (for new disc files).
- Constructing the control blocks required by MPE/3000 for this particular access of the file. The information in these blocks is derived by merging specifications from four sources, listed below in descending order of precedence:
 1. The file label, obtainable only if the file is an *old* file on disc. This information overrides that from any other source. (Label formats are presented in Appendix F.)
 2. The parameter list of a previous :FILE command referencing the same formal file designator named in this FOPEN call, if such a command was issued in this job or session. This information overrides that from the two sources listed next.

3. The parameter list of this FOPEN intrinsic call.
4. System default values provided by MPE/3000 (when values are not obtainable from the above three sources).

When information in one of these four sources conflicts with that in another, preempting takes place according to the order of precedence shown above. To determine the specifications actually taking effect, the user can call the FGETINFO intrinsic, described later in this section. Notice that certain sources do not always apply or convey all types of information. (For instance, no file label exists when a new file is opened and so all information must come from the last three sources above.)

Files On Non-sharable Devices

When a process opens a disc file, the user specifies whether the file is an old or new file; an old file is an existing, labeled file, and a new file implies that the file is to be created. When a process accesses a file that resides on a non-sharable device, the device's attributes may override the user's old/new specification. Specifically, devices used for input only (such as card readers) automatically imply *old* files; devices used for output only (such as line printers) automatically imply *new* files; serial input/output devices (such as teletype terminals and magnetic tape units) follow the user's old/new specifications.

When a job attempts to open an *old* file on a non-sharable device, MPE/3000 searches for the file in the Virtual Device Directory (VDD). If the file is not found, a message is transmitted to the console operator, asking him to locate the file by taking one of the following steps:

1. Indicate that the file resides on a device that is not in auto-recognition mode. No :DATA command is required—the operator simply allocates the device.
2. Make the file available on an auto-recognizing device, and allocate that device.
3. Indicate that the file does not exist on any device; the user's FOPEN request will be rejected.

When a job opens a *new* file on a non-sharable device (other than magnetic tape), the operator is not required to intervene. In these cases, the first available device is used. (A non-sharable device is considered directly available if it is not being used, or if it is being used by the requesting job and is requested by its logical device number.)

The specification of a device class when FOPEN is issued implies a request for the initial allocation of a previously unopened device. (The device parameter is ignored when \$STDIN[X] and \$STDLIST are opened.) A job may reallocate an opened device by specifying the device's logical device number when the file is opened. The FGETINFO intrinsic should be used to determine the logical device number assigned to an opened file.

Multiple processes can asynchronously interleave accesses to reallocated devices. Since the file system does "anticipatory reads" on buffered input devices, multiple processes should set the Inhibit Buffering aoption parameter of the FOPEN intrinsic, if records must be transmitted in the same order as requested.

When a job opens a *new* file on a magnetic tape unit, operator intervention is always required; the operator must make the tape available.

Devicefile Management

A devicefile is a file whose immediate origin or destination is a non-sharable device, i.e., a serial device, which can be accessed by only one job or session at a time. (For this discussion, “jobs” will be understood to cover both jobs and sessions, unless specifically noted otherwise.)

The console operator controls and monitors a device using its logical device number (ldev), assigned during device configuration. MPE retains the correspondence between an input devicefile and the ldev of its non-sharable origin. Similarly, the destination ldev or device class of an output devicefile is recorded.

Control and monitoring of a devicefile is through its unique devicefileid, which has the following form:

$$\left\{ \begin{array}{l} \#Innn \\ \#Onnn \end{array} \right\}$$

I indicates an input devicefile; *O* indicates an output devicefile; and *nnn* is a unique number assigned to the devicefile when it first becomes known to MPE.

Additional information is associated with each devicefile to assist in its identification and management: job name (if supplied by the user), user name, account name, and file name (if supplied), state and associated device are among the attributes recorded for each devicefile. Devicefile attributes can be displayed with the :SHOWIN and :SHOWOUT commands.

INPUT DEVICEFILES. There are three types of input devicefiles: job input devicefiles: :DATA devicefiles, and operator-assigned devicefiles.

A :JOB or :HELLO devicefile becomes the job input file when the job begins execution. There is exactly one such job input file for each job: a reference to \$STDIN(X) by any process of the job is a reference to the job input devicefile. Job input devicefiles can only be submitted on devices designated as job-accepting; this is a device configuration parameter that can be changed dynamically by the console operator (see =ACCEPT/ =REFUSE in the Console Operator's Guide).

In order to automatically direct input data (separate from the job input file) to a particular job for processing, the user provides a :DATA command as the first image of the file. The format of the :DATA command is:

$$:DATA \underbrace{[jobname,]username[/upass].acctname[/apass]} [;filename]$$

jobid

This command permits only a job named *jobid* to access the data in this devicefile, *filename* can be used as a further qualifier to distinguish between multiple :DATA devicefiles directed to a single job. :DATA devicefiles can only be submitted on input devices configured (or changed by the operator) to be data-accepting. :DATA devicefiles should be terminated with a :EOD command.

To permit the operator to control the allocation of input data, i.e., to submit the data on a non-accepting (neither job- nor data-accepting) device, an input devicefile begins simply with the user's first data record. When a job requires such a devicefile, the operator is requested to assign an appropriate, non-accepting device if (or as soon as) one is available.

DEVICE RECOGNITION. Device recognition is a technique whereby MPE will attempt automatic identification of a job input or :DATA devicefile. When an unowned, accepting device is readied, the device recognition function will scan for a legitimate :JOB or :HELLO command (if the device is job-accepting), or a :DATA command (if the device is data-accepting).

A recognized (non-spooled) devicefile will be placed in the READY state. A READY input devicefile is ready for initial allocation to the job, without requiring operator intervention. The =DELETE console command can be used to delete READY :DATA devicefiles; =ABORTJOB will automatically delete the job input devicefile.

A devicefile input on an accepting magnetic tape drive must be contained in a single volume.

OUTPUT DEVICEFILES. Each job-accepting input device is assigned a default output device during device configuration. (Note that the default output device may be an ldev or a device class.) For each job submitted on a given job input device, the corresponding default output device is used to assign a non-sharable destination device for the job list devicefile. The user can override this default by specifying the OUTCLASS parameter on the :JOB card.

There is exactly one such job list file assigned to each executing job. Each reference to \$STDLIST by any process in the job is a reference to the same devicefile.

A job may create other output devicefiles by directing them to a non-sharable ldev (different from the job list device) or to a device class.

CLASSIFICATION OF DEVICEFILES. In summary, devicefiles are defined as follows:

INPUT DEVICEFILES — Input files submitted on non-sharable devices.

Job Input Devicefiles — Always auto-recognized on a job-accepting device. Allocated to job by job-dispatching; file name is \$STDIN[X]. Deallocated at job termination.

:DATA Devicefiles — Always auto-recognized on a data-accepting device. Allocated to specified job on initial request.

Operator-Assigned Devicefiles — When initially requested by job, operator is asked to identify an available, non-accepting device. May not be spooled.

OUTPUT DEVICEFILES — Output files directed to non-sharable devices.

Job List Devicefiles — Allocated by job-dispatching; filename is \$STDLIST. Destination is the default output device, or user OUTCLASS. Deallocated at job termination.

Other Output Devicefiles — Allocated by non-\$STDLIST requests, with destination specified by user. If magnetic tape, operator will specify available device.

DEVICEFILE ALLOCATION. Access to a devicefile is achieved by specifying a non-sharable device or device class in the device parameter of the FOPEN intrinsic. A successful FOPEN results in the allocation of the devicefile to the job. The initial allocation of a devicefile causes it to be placed in the OPENED state. Operator intervention is always required for the initial allocation of devicefiles on non-accepting input devices, and on output magnetic tapes. If an OPENED devicefile is not spooled, the device on which it resides is owned by the job until the final FCLOSE of the file.

The specification of a device class at FOPEN-time implies a request for the initial allocation of a previously unopened devicefile. (The device parameter is ignored for FOPEN's of \$STDIN[X] and \$STDLIST.) A job may re-allocate an OPENED devicefile by specifying the devicefile's ldev at FOPEN-time: the FGETINFO intrinsic should be used to determine the ldev assigned to an OPENED file. When an OPENED devicefile is re-allocated, its initial file name is retained; the file name supplied for the re-allocation is ignored.

In general, when a process FOPEN's a disc file, it specifies whether the file is an OLD or NEW file. OLD implies the file existed prior to the request; NEW implies that the file is being created by the request. This distinction is maintained when specifying input/output access on devicefiles, with the refinement that the device's attributes may override the user's declaration. Specifically, input-only devices (e.g., card readers) automatically imply input access; output-only devices (e.g., line printers) automatically imply output access; user's OLD/NEW specifications take effect for input/output devices opened for input/output access (e.g., teletypes, mag tapes).

It is possible for multiple processes to asynchronously interleave accesses to re-allocated devicefiles. Since the file system does "anticipatory reads" on (non-spooled) buffered input devicefiles, multiple processes should set the Inhibit Buffering AOPTION if records have to be transmitted in the same order as requested.

INPUT DEVICEFILE ALLOCATION. For the initial allocation of an input devicefile, MPE tries to locate a READY :DATA file. For such a devicefile to be selected, it must have been recognized on the device or class specified at FOPEN-time, and its *jobid* must match exactly that of the requesting job. However, any job with a given *jobid* can allocate a :DATA file with the same *jobid*. To prevent unintended allocations, users are cautioned to provide unique *jobid*'s for those jobs accessing :DATA devicefiles.

The search for the READY :DATA file proceeds as follows. If a name is supplied at FOPEN-time, an attempt is made to find a devicefile with a *filename* matching that name; if none is found, a devicefile with a null *filename* will satisfy the request. If no name was specified, an attempt is made first to find a devicefile with a null *filename*; if this fails, any READY devicefile found, which came from the specified origin and has the correct *jobid*, will satisfy the request.

If the search for a READY :DATA devicefile fails and an ldev was specified in the device parameter, an attempt is made to find and re-allocate any OPEN devicefile (allocated to this job) which came from the specified ldev, regardless of filename.

If an appropriate input devicefile is not found, the operator is asked to "locate" the file by:

1. Indicating that it is on a non-accepting device. No :DATA card is (i.e., can be) supplied, and the device is allocated for the job's exclusive use.
2. Indicating that it has just been recognized on a data-accepting device. The first image is a :DATA command.
3. Indicating that the file is on no device. The FOPEN request will be rejected.

OUTPUT DEVICEFILE ALLOCATION. If a non-tape device class is specified in the FOPEN of an output devicefile, MPE directs the devicefile to the first unowned device in the class. If there are no unowned devices in the class, but at least one of them is spoofled, MPE will defer the assignment of a real device until the completed devicefile is scheduled for output spooling; in this case, device-related information returned by FGETINFO will be representative of the class. (Actually, attributes of the first device in the class are returned.)

If a non-tape ldev is specified in the device parameter, access is granted if the device is available. A non-sharable device is considered to be available if it is unowned, or spoofled, or owned by the requesting job and the request is by ldev (i.e., re-allocation).

If a magnetic tape ldev or device class has been specified, the console operator will be asked to assign the device:

1. If the user has requested a tape device class, the operator may specify any unowned device in the class.
2. If the user has requested the ldev of a specific tape drive, the operator may assign the tape if it is available.

DEVICE-RELATED FEATURES. MPE provides special services to assist the operation of card punches and line printers.

Spooling is a technique designed to alleviate these problems. Under console operator control, spooling buffers entire devicefiles on disc. User access to the devicefile is directed automatically to the disc copy, which is called a spooled devicefile, or spoofle. Since disc is a sharable device, jobs can concurrently access multiple spoofles which came from (or are destined for) the same non-sharable device. Furthermore, the spooling (i.e., 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 "transparent" to the user. This means that the user's job is unaware that it is accessing a spoofle instead of a real devicefile on a non-sharable device. 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; e.g., reading hardware status.)

There are, however, additional features made available to the user by MPE's capability to spool devicefiles. These include:

- The `:STREAM` command, enabling a job to introduce a new `:JOB` or `:DATA` devicefile; an ASCII file on any device can be "spooled" in this manner. The availability of this facility is controlled by the console operator.
- The ability to automatically generate multiple copies of spooled output devicefiles.
- The recovery of spooled devicefiles in the event of a system failure.

Overview of Operation. Certain devices will automatically be spooled when MPE is cold-loaded. The console operator can also initiate spooling on an unowned, non-sharable device. This causes the device to be allocated to a spooler system process; a device owned by a spooler is called a spolee. The console operator also terminates a spooler and returns its device to the unowned state.

Input spoolers may be started on those card readers or magnetic tape units designated as job-accepting devices. Output spoolers may be assigned to line printers and card punches.

During the spooling operation, a spoofle and the spooler copying (spooling) it are both in an ACTIVE state. ACTIVE spoofles are not available for access by a job. When an input devicefile has been completely spooled, the resultant spoofle is placed in the READY state, available for access.

As discussed previously (Devicefile Allocation), when a program is accessing a devicefile, the devicefile is in the OPENED state. In the last FCLOSE of an output spoofle, the file is changed from the OPENED to the READY state. Output spoolers choose READY, output spoofles for processing. (A READY output spoofle cannot be accessed by a user job or session.) When an output spooler finishes a copy of an output spoofle, if more copies must be made, the spoofle is changed from the ACTIVE to the READY state and re-scheduled for output. Figure 6-1 summarizes input and output devicefile states.

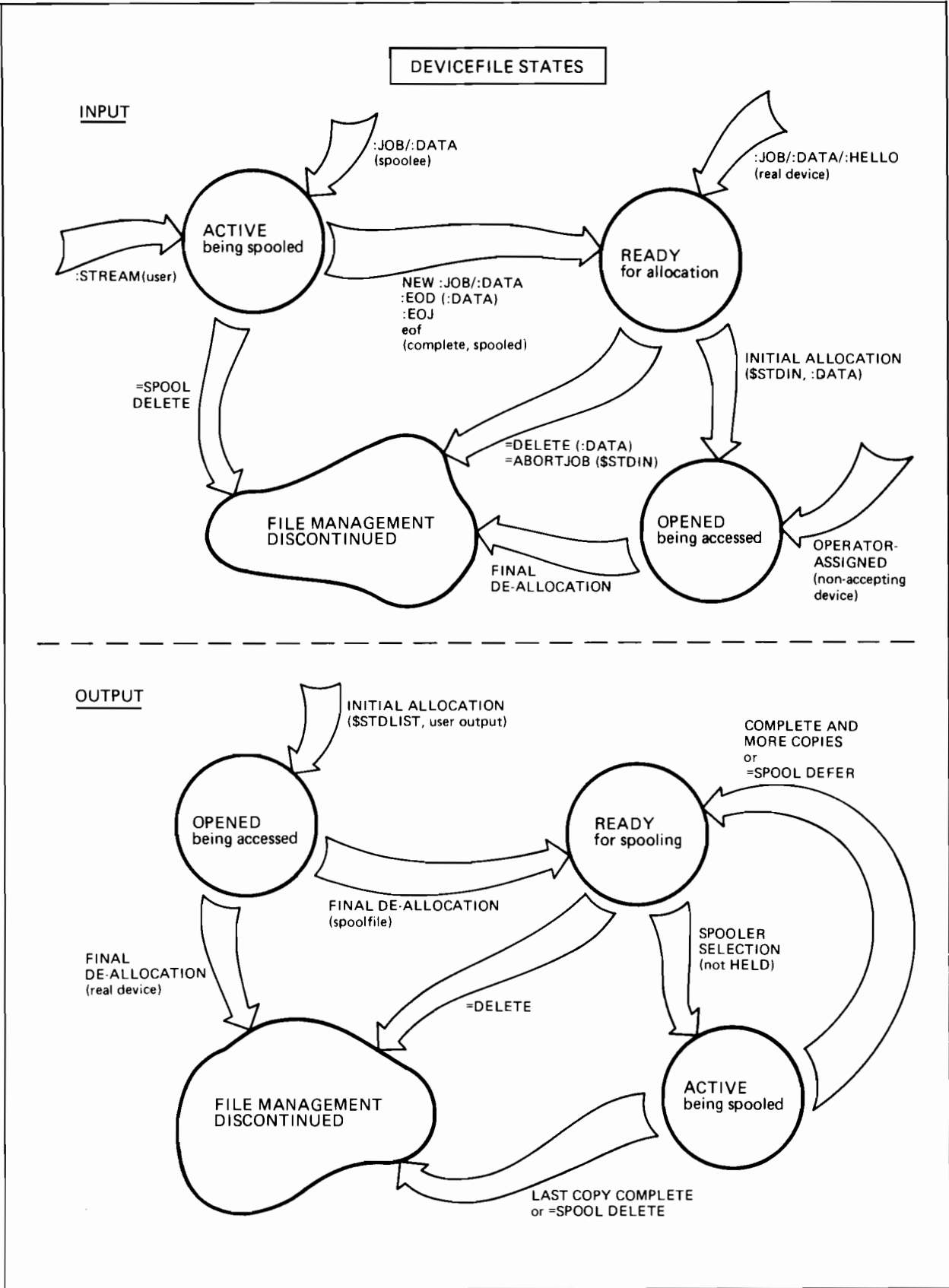


Figure 6-1. Devicefile States

The description of devicefile management applies equally to spooled devicefiles. Spoolfiles reside on spooling discs, which are discs included in the special device class "SPOOL". A spooling disc is, of course, not excluded from containing file system files. The total disc space devoted to spoolfiles can be limited during system configuration. The maximum number of concurrently OPENED spoolfiles is also a configuration parameter.

Figure 6-2 presents an overview of devicefile operation, including spooling. Details of spooling operation are described in the following paragraphs.

Input Spooling. An input spooler initiates a device recognition function similar to the one described under Devicefile Management. Upon recognition of a :JOB image (if the spooler is job-accepting) or :DATA image (if the spooler is data-accepting), the input spooler copies the subsequent records into an ACTIVE spoolfile. When the spooler encounters a :EOD command (if a :DATA devicefile is ACTIVE), a :EOJ, a physical end-of-file, or a new :JOB or :DATA command, the ACTIVE spoolfile is placed in the READY state, and the recognition function is repeated.

When there are no more devicefiles on the device, the spooler enters a WAITING state, waiting for more input.

The console operator can explicitly place the spooler in a WAITING state. On receiving this command, the spooler will normally finish the current ACTIVE spoolfile, if any, and then wait for another operator command. However, the operator may specify that the ACTIVE spoolfile should be DELETED and the wait state entered immediately.

Likewise, when the operator terminates a spooler, any currently ACTIVE spoolfile will be completed before the spooler terminates, unless the operator also specifies the DELETE option.

Spoolfile Access. The devicefile allocation procedures described under Devicefile Management apply to the allocation of spoolfiles. A request for the [re-] allocation of an input devicefile which has been spooled results in access to the appropriate input spoolfile, MPE's Input/Output System (I/O system) directs the job's input requests to the spoolfile, retrieving the appropriate records from disc in a manner transparent to the user.

(Extents of :JOB and :DATA spoolfiles are released as they are accessed, if (and only if), the accessing job is not RESTARTable, as specified in the :JOB command. Omission of this parameter, therefore, will minimize disc occupancy by very large input spoolfiles.)

An FOPEN request for a NEW devicefile directed by ldev to an output spooler will result in access to an output spoolfile. An FOPEN request for a NEW devicefile directed to a device class will result in a new spoolfile being created, if there is at least one spooler in the class and there are no unowned devices in the class. (However, even if there is an unowned device in the class, a new spoolfile will be created if multiple copies were requested in the FOPEN, and there is at least one spooler in the class.)

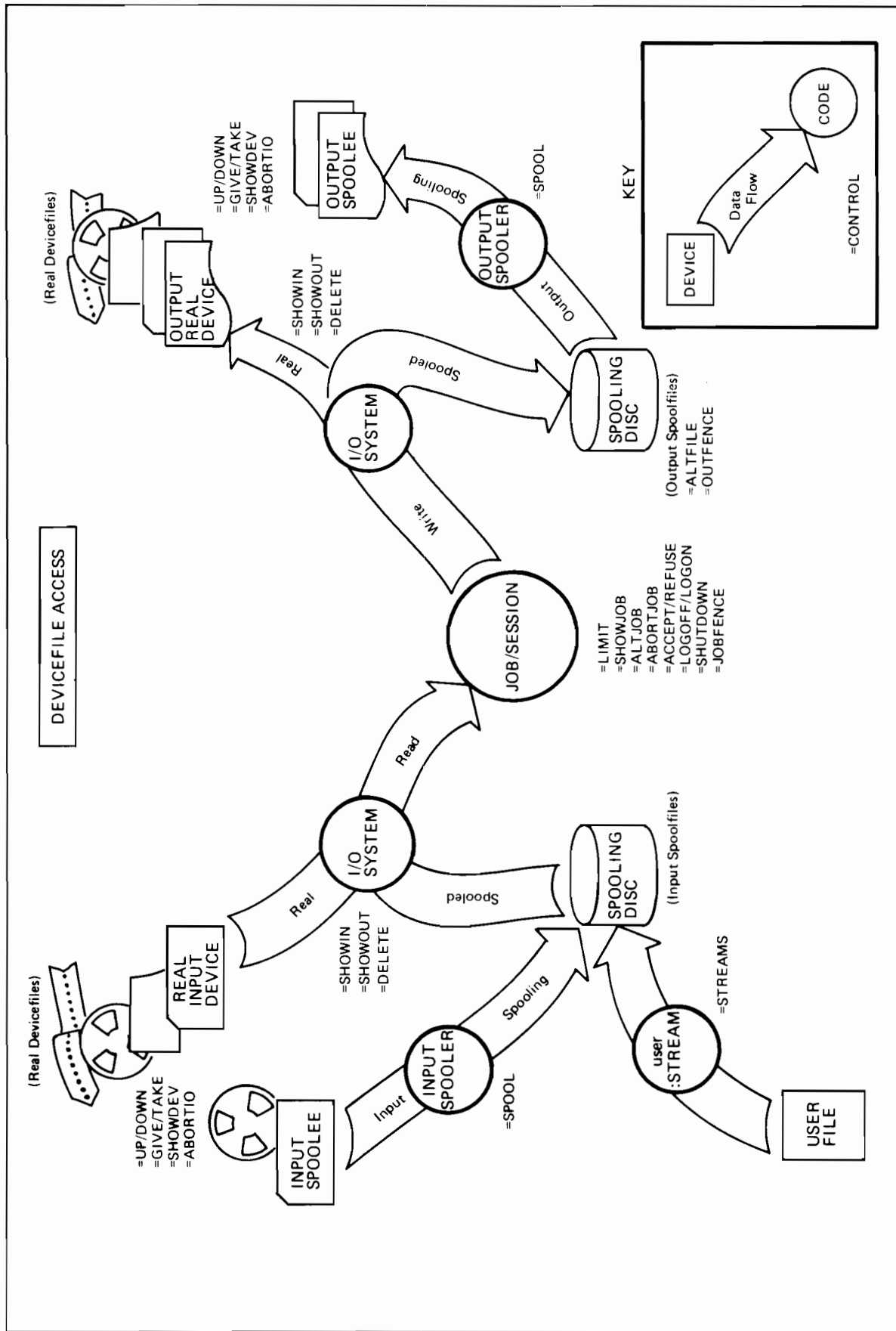


Figure 6-2. Devicefile Access

User write requests to spooled output devicefiles are diverted by the I/O system to the appropriate spoofer, again in a manner transparent to the user. An OPENED, output spoofer is placed in the READY state on the final FCLOSE of the file.

If necessary, a READY output spoofer can be deleted by the console operator.

Output Spooling. An output spooler searches for a READY output spoofer directed to its spooler, or to a device class of which its spooler is a member. Different spoolers destined for the same device are copied on a priority basis, as determined by the file's *outputpriority* parameter; selection is FIFO for spoolers with the same *outputpriority*. In addition, the console operator can set an output fence. Output spoolers with *outputpriority* less than or equal to the current output fence are deferred; deferred spoolers are not selected for spooling by output spoolers. Note that an *outputpriority* of 0 is always less than or equal to the fence. When an output spooler copies a spoofer, it produces the header, trailer, and FORMS dialogue, like real device access.

An output spooler which fails to find a spoofer to copy will enter a WAITING state. It will automatically be reactivated when a spoofer directed to it becomes READY for copying.

The operator can change the *outputpriority*, destination, and number of copies of an OPENED or READY output spoofer.

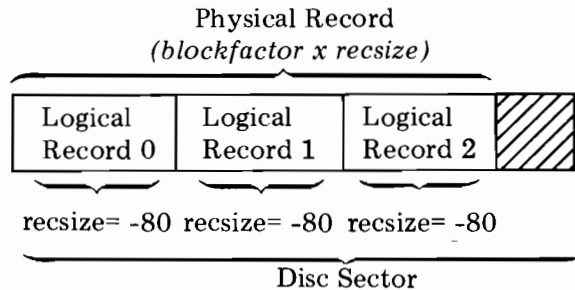
To minimize disc occupancy, extents of output spoolers are released as the final (possibly the only) copy is being made. An interruption (e.g., =SPOOL DEFER or a system failure) of the last copy will cause the remaining extents to be copied when the spoofer is re-scheduled. In such cases there will always be some overlap with the previous output, so that no data is lost. Interruption of other than the last copy will, however, cause the entire copy to be output from the beginning upon re-scheduling.

When a spoofer with FORMS currently on the device is interrupted (by =SPOOL DEFER or a system failure), an indication that special FORMS are required will be issued to the console operator, when the spoofer is again selected for spooling. In these cases, there will always be sufficient overlap with the previous output so that alignment of the FORMS can be re-established visually/manually, using the overlapped lines.

Record Formats

A file can contain records written in one of three formats: *fixed-length*, *variable-length*, and *undefined length*. These formats are described below. (In all cases, *reclsize* is in words.)

For *fixed-length records*, physical records are blocks containing one or more logical records. The block size is determined by multiplying the block factor by the logical record size. (The block factor and logical record size are specified in the *blockfactor* and *reclsize* parameters of the FOPEN intrinsic.) On any one file, fixed-length records are all the same size. A 128-word physical record (block) containing three 80-byte, fixed-length logical records is illustrated below:

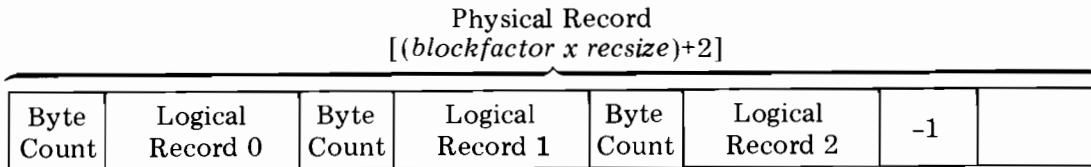


For *variable-length records* (as for fixed-length records), physical records are blocks containing one or more logical records--but, on any one file, the record size may vary from record to record. The block size is determined by multiplying the blockfactor by the *reclsize* parameter specified by the user, and adding two words (reserved for file-system use).

$$\text{Actual Blocksize} = (\text{blockfactor} \times \text{reclsize}) + 2$$

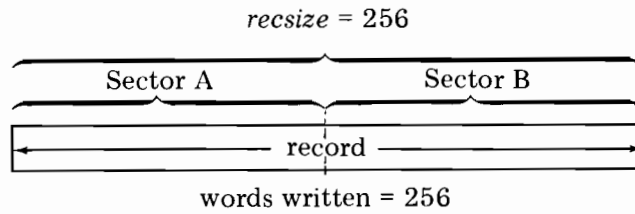
(in words)

In a block containing variable-length records, 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 "-1", acting as the *block terminator*; the next logical record encountered will be the first record in the next block. The block format is:

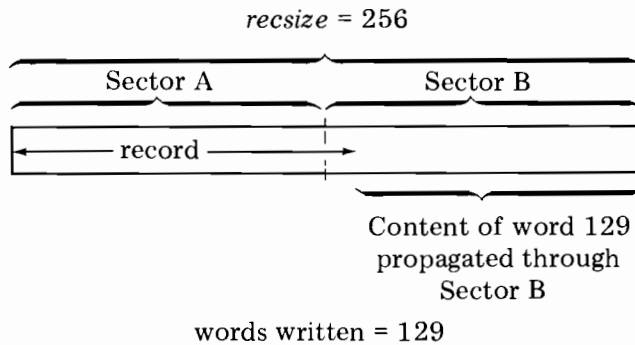


For *undefined-length records*, physical records and logical records are synonymous--that is, Physical Record A is the same as Logical Record A. For records of this type, the *reclsize* parameter specified by the user denotes the size of the longest record to be transferred. The format of undefined records written to disc, with respect to the disc sectors occupied, can be illustrated by three cases in which the user-specified *reclsize* is 256.

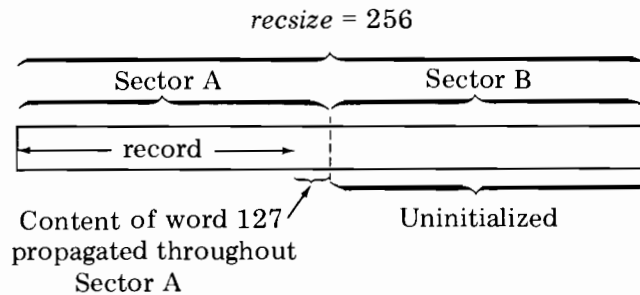
Case 1: The user writes a record 256 words long. The full record completely fills two disc sectors.



Case 2: The user writes a record 129 words long. The record written occupies all of Sector A and the first word of Sector B; the last word written is propagated throughout the remainder of Sector B. (The rule is: if (*reclength*) modulo 128 is not zero, then the last word written is propagated through the current sector.)



Case 3: The user writes a record 127 words long. The record written occupies 127 words of Sector A; the last word of the record is propagated throughout the remainder (word 128) 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).)



The FOPEN intrinsic call is written in this format:

| | |
|--------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>INTEGER PROCEDURE</i> | <i>FOPEN</i> (formaldesignator,foptions,aoptions,resize,device,formmsg,userlabels,blockfactor,numbuffers,filesize,numextents,initialloc,filecode); |
|--------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------|

VALUE foptions,aoptions,resize,userlabels,blockfactor,numbuffers, filesize,numextents,initialloc,filecode;

BYTE ARRAY formaldesignator,device,formmsg;

LOGICAL foptions,aoptions;

INTEGER resize,userlabels,blockfactor,numbuffers,numextents,initialloc, filecode;

DOUBLE filesize;

OPTION VARIABLE, EXTERNAL;

* See FSET in FORTRAN manual for entering this new file in the proper place in the FILE table. (p.8-7)
 CCE = 01K
 CCL = 0FC
 CCG IS NOT RETURNED

This intrinsic returns (as the value of FOPEN) an integer file number used to identify the opened file in other intrinsic calls, (and changes the condition code in the status register as noted later in this discussion).

The FOPEN intrinsic parameters specify the elements shown below.

formaldesignator A byte-array containing a string of ASCII characters, interpreted as a formal file designator (as defined in Section V). This string must begin with a letter, contain alphanumeric characters, slashes, or periods and terminate with any non-alphanumeric character except a slash or a period. (If the string names a system-defined file, it can begin with a dollar sign (\$); if it names a user-predefined file, it can begin with an asterisk (*).) This parameter can be omitted; the default value assigned is a temporary nameless file that can be read or written on, but not saved.

foptions A 16-bit value that denotes a combination of file characteristics, including the type, recording code, and record format of the file. The meaning of the bit groups, and the way the user sets them are illustrated later. This parameter can be omitted; as a default value, all bits are set to zero.

aoptions A 16-bit value that denotes a combination of access options associated with the file. These options include restrictions on reading or writing on the file, or exclusive access provisions. The meaning of the bit groups and the way the user sets them are illustrated later. If this parameter is omitted, all bits are set to zero.

recsize

An integer indicating the size of the logical records in the file. If a positive number, this represents words; if a negative number, this indicates bytes. If the file is a new file, this value is permanently recorded in the file label. If the records in the file are of variable length, this value indicates the maximum logical record length allowed. For unit-record devices, the default value is the physical record width of the associated device; for all other devices, the default value is 128 words.

BINARY files are word oriented. A record size specifying an odd byte count is rounded up by FOPEN to the next highest even number.

ASCII files may be created with LOGICAL records which are an odd number of bytes in length. Within each block; however, records begin on word boundaries.

For either BINARY or ASCII files which are fixed or undefined format, it is the rounded record size, i.e., $(recsize+1)$ and LSR(1), which should be used in computations which also include *blockfactor* or block size.

device

A byte array containing a string of ASCII characters terminating with any non-alphanumeric character except a slash or period, designating the device on which the file is to reside. The string may represent a device class name (up to eight alphanumeric characters beginning with a letter) or a logical device number (a three-byte numeric string) as described under the *device* parameter for the :FILE command. The default value is a byte-array containing the string DISC. (Device class names and logical device numbers are defined and assigned to devices during system configuration.)

formmsg

← 50
char

A byte array containing a forms message that can be used for purposes such as telling the console operator what type of paper to use in the line printer. This message must be displayed to the operator and verified before this file can be printed on a line printer. The message itself is a string of ASCII characters terminated by a period. If this parameter is omitted, no forms message is available. MAXLEN = 45 CHAR

userlabels

An integer specifying the number of user-label records to be written for this file. The default number is 0. CT'75

blockfactor

The size of each buffer to be established for the file, specified as an integer equal to the number of logical records per block. (For fixed-length records, *blockfactor* is the actual number of records in a block. For variable-length records, *blockfactor* is interpreted as a multiplier used to compute the block size (maximum $recsize \times blockfactor$). For undefined-length records, *blockfactor* is always one logical record per block. The *blockfactor* value specified by the user may be overridden by MPE/3000. The default value is calculated by dividing the specified *recsize* into the configured physical record size; this value is rounded downward to an integer, but is never less than 1. Specification of a negative or zero value results in the default *blockfactor* setting. Valid range is 1 through 255. Values greater than 255 are defaulted to *blockfactor* modulo 256.

numbuffers

NOTE: Bits groups are denoted using the standard SPL/3000 notation. Thus Bits (14:2) indicates Bits 14 and 15; Bits (10:3) indicates Bits 10, 11, and 12.

| Bits | Meaning |
|--------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| (0:4) | <u>An integer which, for spooled output devices, specifies the outputpriority to be attached to this file.</u> This priority is used to determine the order in which files will be produced, when several are waiting for the same device. This must be a number between 1 (lowest priority) and 13 (highest priority), inclusive. If this value is less than the current output fence set by the console operator, the file will be deferred from printing/punching until the operator raises the <i>outputpriority</i> of the file or lowers the output fence. This parameter can be specified for a file already FOPENED (e.g., \$STDLIST), in which case the highest value supplied before the last FCLOSE will take effect. This parameter is ignored for non-spooled output devices. <u>The default value is 8.</u> |
| (4:7) | <u>An integer which, for spooled output devices, it specifies the number of copies of the entire file to be produced by the spooling facility.</u> This can be specified for a file already FOPENed (e.g. \$STDLIST), in which case the highest value supplied before the last FCLOSE will take effect. The copies will not appear contiguously if the console operator intervenes or if a file of higher <i>outputpriority</i> becomes READY before the last copy is complete. This parameter is ignored for non-spooled output devices. <u>The default value is 1.</u> |
| (11:5) | <u>An integer specifying the number of buffers to be allocated to the file. This parameter is not used for files representing interactive terminals, since a system-managed buffering method is always used in such cases. If omitted, or set to zero or a negative number, the default value is 2.</u> |

filesize

A double-word integer (as defined in SPL/3000) specifying the maximum file capacity in terms of physical records (for files containing variable-length and undefined-length records) and logical records (for files containing fixed-length records). The default value is 1023. A zero or negative value results in the default *filesize* setting. The maximum capacity allowed is over one million (2^{20}) sectors. (The number of sectors in a file is found by the formula shown under FILE CHARACTERISTICS in Section V.) However, since all extents must reside on one volume, the largest *filesize* currently available is limited to the 184,000-sector capacity of the HP 2888A Disc File.

- numextents** An integer specifying the 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 is determined by the *filesize* parameter value divided by the *numextents* parameter value. If specified, *numextents* must be an integer from 1 to 16. The default is 8. A zero or negative value results in the default *numextents* setting.
- initalloc** An integer specifying the number of extents to be allocated to the file when it is opened (rather than dynamically, as needed). This must be an integer from 1 to 16. The default value is one. If an attempt to allocate the requested space fails, the FOPEN intrinsic returns an error condition code to the user's program.
- filecode** An integer recorded in the file label and made available for general use to anyone accessing the file through the FGETINFO intrinsic call, described later. This parameter is used for new files only. For this parameter, any user can specify a positive integer ranging from 0 to 1023. (The default value is 0.) If a user's process is running in privileged mode, the user can specify a negative integer for *filecode* upon initially opening a file. Then, any future accesses of the file must be requested in privileged mode and must also specify the correct *filecode*. Certain positive integers beyond 1023 have particular HP-defined meanings

These special integers are:

| Integer | Meaning |
|---------|---------------------------------------|
| 1024 | A USL file. |
| 1025 | A BASIC/3000 data file. |
| 1026 | A BASIC/3000 program file. |
| 1027 | A BASIC/3000 fast program file. |
| 1028 | A relocatable library (RL) file. |
| 1029 | A program file. |
| 1030 | A STAR/3000 file. |
| 1031 | A segmented library (SL) file. |
| 1040 | A Cross Loader ASCII file (SAVE). |
| 1041 | A Cross Loader relocated binary file. |
| 1042 | A Cross Loader ASCII file (DISPLAY). |
| 1050 | An EDIT/3000 KEEPQ file (non-COBOL). |
| 1051 | An EDIT/3000 KEEPQ file (COBOL). |
| 1060 | A RJE punch file. |

Foptions Parameter

The foptions parameter allows the user to specify six different file characteristics, by setting corresponding bit groupings in a 16-bit word. The correspondence is from right to left, beginning with Bit 15. If this parameter is omitted, all bits are set to zero. These characteristics are as follows, proceeding from the rightmost bit groups to the leftmost bit groups in the word. (The bit settings are also summarized, for the convenience of the user, in Figure 6-3.)

NOTE: Bits groups are denoted using the standard SPL/3000 notation. Thus Bits (14:2) indicates Bits 14 and 15; Bits (10:3) indicates Bits 10, 11, and 12.

Bits

Characteristics

(14:2) *Domain Foption.* The file domain to be searched by MPE/3000 to locate the file, indicated by these bit settings:

- 00 = The file is a *new* file, created at this point. No search is necessary.
- 01 = The file is an old permanent file, and the system file domain should be searched.
- 10 = The file is an old temporary file, and the job file domain should be searched.
- 11 = The file is an old file that is to be located by first searching the job file domain and then, if the file is not found, by searching the system file domain.

OLD TEMP

(13:1) *ASCII/BINARY Foption.* The code (ASCII or binary) in which a *new* file is to be recorded when it is written to a device that supports both codes. In the case of disc files, this also effects padding that can occur when a direct-write intrinsic call (FWRITEDIR) is issued to a record that lies beyond the current logical end-of-file indicator. In ASCII files, any dummy records between the previous end-of-file and the newly-written record are padded with blanks; in binary files, such records are padded with binary zeros. (All files not on disc or tape are treated as ASCII files.) For ASCII, this bit is 1; for binary, it is 0. The default value is 0.

(10:3) *Default File Designator Foption.* The *actual* file designator equated with the formal file designator specified in FOPEN, if

1. No explicit or implicit :FILE command equating the formal file designator to a different actual file designator is encountered in the job or session; or
2. The *Disallow File Equation Foption* (below) is specified (5:1)

The bit settings are

- 000 = The actual file designator is the same as the formal file designator.
- 001 = The actual file designator is \$STDLIST.
- 010 = The actual file designator is \$NEWPASS.
- 011 = The actual file designator is \$OLDPASS.
- 100 = The actual file designator is \$STDIN.
- 101 = The actual file designator is \$STDINX.
- 110 = The actual file designator is \$NULL.

Bits

Characteristics

(8:2)

Record Format Foption. The format in which the records in the file are recorded, indicated by these bits:

- 00 = Fixed length records; the file is composed of logical records of uniform length.
- 01 = Variable-length records; the file contains logical records of varying length. This format is restricted to sequential access only. The records are written sequentially and the size of each is recorded internally. (The actual record size used is determined by multiplying the *recsize* (specified or default) by the *blockfactor*, and adding two words reserved for system use. This option is not allowed when NOBUF is specified. In such a case, the record format used is *undefined-length records*, discussed below.)
- 10 = Undefined-length records. The file contains records of varying length that were not written using the variable-length foption (01) just described. The undefined-length foption is typically used when reading a magnetic tape written under another operating system. All files not on disc or tape are treated as containing undefined-length records.

(7:1)

Carriage Control Foption. If selected, this specifies that the user will supply a carriage-control character in the calling sequence of each FWRITE call that writes records onto the file.

- 1 = Carriage-control character expected.
- 2 = No carriage-control character expected.

OCT 75

Carriage control will be defined only for character oriented, i.e. ASCII files. This option and BINARY will be considered as mutually exclusive by FOPEN, and attempts to open new files with both BINARY and this option specified will result in an access violation. Since this option is a physical attribute of the file, its state cannot be modified when opening an OLD disc file.

A carriage control character passed through the *control* parameter of FWRITE will be recognized and acted upon only for files for which carriage control is specified. Therefore, embedded control will be treated strictly as data for files for which no carriage control is specified, and will not invoke spacing for such files. Users may specify spacing action on carriage control-specified files, either by embedding the control in the record, indicated with a *control* parameter of one in the call to FWRITE, or by sending the control directly via the *control* parameter of FWRITE.

A carriage control character sent to a file on a device on which the control cannot be executed directly, e.g., line spacing to a disc or tape file, will result in having the control character embedded as the first byte of the record. Thus, the first byte of each record in a disc file having a carriage control character will be control information. Control sent to other types of files results in transmission of the control to the driver.

Bits

Characteristics

(7:1)
(cont.) The control codes %400 through %403 will be remapped to %100 through %103 so that they will fit into one byte and thus can be embedded. Records written to the line printer with one of the above controls should not contain information other than control information. In order that the FWRITE of such a control record act the same as the equivalent call to FCONTROL, the printer driver will execute only the control portion of the record and will not transfer any other data included in the record.

For the purpose of computing record size, carriage control information will be considered by the file system as part of the data record. As such, specifying the carriage control option will add one byte to the record size at the time the file is created; i.e. a specification of REC = -132, 1,F,ASCII; CCTL will result in a *resize* of 133 characters.

A user may always read up to and including the *resize* as returned by FGETINFO. On writes of files for which carriage control is specified, however, the data transferred is limited to *resize*-1 unless a control of one is passed indicating the data record is prefixed with embedded control.

(6:1) This bit is reserved for system use.

(5:1) *Disallow File Equation.* This option ignores any corresponding :FILE command, so that the specifications in the FOPEN call take effect (unless preempted by those in the file label.) For disallowing :FILE, the bit is set to 1; for allowing :FILE, the bit is set to 0. The default value is 0.

(0:5) These bits are reserved for system use.

| BITS | (0:5) | (5:1) | (6:1) | (7:1) | (8:2) | (10:3) | (13:1) | (14:2) |
|---------|----------|---------------------------|----------|------------------------|-----------------------------------------------|----------------------------------------------------------------------------------------------------------------------------|------------------------------|------------------------------------------------------------------------------------|
| FIELD | (Unused) | Disallow :FILE | (Unused) | Carriage Control | Record Format | Default Designator | ASCII/ Binary | Domain |
| MEANING | | 1 = No :FILE 0 = :FILE | | 0 = NOCCTL 1 = CCTL | 00 = Fixed 01 = Variable 10 = Undefined | 000 = filename 001 = \$STDLIST 010 = \$NEWPASS 011 = \$OLDPASS 100 = \$STDIN 101 = \$STDINX 110 = \$NULL | 0 = Binary 1 = ASCII . | 00 = New file 01 = Old System File 10 = Temporary File 11 = Old User File |

Figure 6-3. Foptions Bit Summary

The programmer can establish the bit settings desired through various SPL/3000 conventions. In one such convention, he enters the octal equivalents of the bit settings as the *foptions* parameter in the FOPEN call (unless he wishes the default foptions assigned). As with all octal values entered in SPL/3000, this parameter must be preceded by a percent sign to denote the octal base.

EXAMPLE:

Suppose that the user wants to specify, for *foptions*, that the system file domain is to be searched for the referenced file (Bits 14-15 = 01), and that the default actual file designator is to be \$STDIN (Bits 10-12 = 100), he would first determine the following complete bit setting, calculate its octal equivalent, and write this (preceded by %) as the *foptions* parameter in the FOPEN intrinsic call.

| | | | | | | | | | | | | | | | | |
|------------------|-----------------------------------------|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
| Bit No. | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| Bit Setting | (Reserved for system use—always zeros.) | | | | | | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| Octal Equivalent | 0 | 0 | | | 0 | | | | 0 | | | 4 | | | 1 | |
| Sign Bit | ↑ | | | | | | | | | | | | | | | |

The value used for the *foptions* parameter is %000041. (The leading zeros are optional.)

Aoptions Parameter

The aoptions parameter permits the user to specify up to five different access options established by bit groupings in a 16-bit word. If this parameter is omitted, all bits are set to zero. These access options are as follows. (The bit settings are also summarized in figure 6-4.)

Bits

Characteristics

- (12:4) *Access Type Aoptions.* The type of access allowed users of this file:
- 0000 = Read-access only. (The FWRITE, FUPDATE, and FWRITEDIR intrinsic calls cannot reference this file.)
 - 0001 = Write-access only. Any data written on the file prior to the current FOPEN request is deleted. (The FREAD, FREADSEEK, FUPDATE, and FREADDIR intrinsic calls cannot reference this file.)
 - 0010 = Write-access only, but previous data in the file is *not* deleted. (The FREAD, FREEDSEEK, FUPDATE, and FREADDIR intrinsics cannot reference this file.)

Bits

Characteristics

- 0011 = Append-access only. The FREAD, FREADDIR, FREAD-SEEK, FUPDATE, FSPACE, FPOINT and FWRITEDIR intrinsic calls cannot be issued for this file. (This option is not valid with files containing variable-length records.)
- 0100 = Input/Output access. Any file intrinsic except FUPDATE can be issued against this file.
- 0101 = Update access. All file intrinsics, including FUPDATE, can be issued for this file.
- 0110 = Execute access. Allows users with *Privileged Mode Capability* Input/Output access to any loaded file.

(11:1) *Multirecord Aoption.* Signifies that individual read or write requests are not confined to record boundaries. Thus, if the number of words or bytes to be transferred (specified in the *tcount* parameter of the read or write request) exceeds the size of the physical record (i.e., block) referenced, the remaining words or bytes are taken from subsequent successive records until the number specified by *tcount* have been transferred. This option is available only if the Inhibit-Buffering Aoption (below) is also selected. For multirecord mode, this bit is set to 1; for non-multirecord mode, it is set to 0. The default value is 0.

(10:1) *Dynamic Locking Aoption.* Indicates that the user wants to use the FLOCK and FUNLOCK intrinsics to dynamically permit or restrict concurrent access to the file by other processes at certain times. The user's process can continue this temporary locking/unlocking until it closes the file. Dynamic locking/unlocking is made possible through a resource identification number (RIN) assigned to the file and temporarily acquired by the FOPEN intrinsic. The calling process must use the RIN in cooperation with other processes also using it to guarantee the integrity of the file, as discussed in Section IX. Non-cooperating processes are allowed concurrent access at all times (unless other provisions prohibit this). The bit settings are

- 1 = Allow dynamic locking/unlocking.
- 0 = Disallow dynamic locking/unlocking.

A file may be multiply-accessed only if all FOPEN requests for the file specify dynamic locking, or if none of them do. An FOPEN request that disagrees with the current access (if any) will fail.

(8:2) *Exclusive Aoption.* This aoption specifies whether a user has continuous exclusive access to this file, from the time it is opened to the time it is closed. This option is often used when performing some critical operation, such as updating the file. The bit settings allow these choices:

- 01 = *Exclusive Access.* After the file is opened, prohibits another FOPEN request, whether issued by this or another process, until this process issues the FCLOSE request or terminates. If any process is already accessing this file when this FOPEN call is issued, a CCL error code is returned to the calling process. If another FOPEN call to this file is issued while the exclusive aoption is in force, an error code is returned to that calling process. The *exclusive access* can only be requested by users allowed the file-locking access mode by the security provisions for the file.

- 10 = *Semi-Exclusive Access*. After the file is opened, prohibits concurrent output access to this file through another FOPEN request, whether issued by this or another process, until this process issues the FCLOSE request or terminates. A subsequent request for the *input/output* or *update* access-type option will obtain read-only access. Other read-accesses, however, are allowed. If any process already has output access to the file when this FOPEN call is issued, a CCL error code is returned to the calling process. If another FOPEN call that violates the read-only restriction is issued while the *semi-exclusive* option is in effect, that call fails and an error code is returned to the calling process. The *semi-exclusive* access can only be requested by users allowed the file-locking access mode by the security provisions for the file.
- 11 = *Share Access*. After the file is opened, permits concurrent access to this file by any process, in any access mode (subject to other basic MPE/3000 security provisions in effect.)
- 00 = *Default Value*, relating to *Access-Type Aoption*. If the *read-access only* access-type option is selected, *share-access* (11) takes effect. Otherwise, *exclusive access* (01) takes effect.

(7:1)

Inhibit-Buffering Aoption. When selected, this option inhibits automatic buffering by MPE/3000 and allows input/output to take place directly between the user's stack or extra data segment and the applicable hardware device.

The bit setting is:

- 1 = To inhibit buffering.
0 = To allow normal buffering.

NOBUF access is oriented around the transfer of physical blocks rather than logical records.

With NOBUF access, the user has responsibility for blocking and deblocking of records in the file. To be consistent with files built using buffered I/O, records should begin on word boundaries, and when the information content of the record is less than the defined record length, the record should be blank, filled if the file is ASCII or zero filled if the file is BINARY.

The *recsize* and block size for files executed under NOBUF access will now follow the same rules as those created using buffering. The default *blockfactor* for a file created under NOBUF is one.

When a NOBUF file is opened without *multirecord* access, the amount of data transferred per read or write is limited to a maximum of one block.

| BITS | (0:7) | (7:1) | (8:2) | (10:1) | (11:1) | (12:4) |
|---------|----------|----------------------|---------------------------------------------------------------------|-----------------------------------------|-----------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| FIELD | (Unused) | Inhibit Buffering | Exclusive Access | Dynamic Locking | Multi-record Access | Access Type |
| MEANING | | 1 = NOBUF 0 = BUF | 01 = Exclusive 10 = Semi-exclusive 11 = Share 00 = Default | 0 = No Dynamic Lock 1 = Dynamic Lock | 1 = Multi-record 0 = No multi-record | 0000 = Read only 0001 = Write only 0010 = Write (save) only 0011 = Append only 0100 = Read/write 0101 = Update 0110 = Execute |

Figure 6-4. Aoptions Bit Summary

Bits

Characteristics

The EOF, next record pointer, and record transfer count are maintained in terms of logical records for all files. The number of logical records affected by each transfer will be determined from the size of the transfer.

Transfers always begin on a block boundary. Those which are not whole blocks leave the next record pointer set to the first record in the next block. The EOF pointer always points at the proper logical record.

For files opened with NOBUF access the FREADDIR, FWRITEDIR, and FPOINT intrinsics will consider the *recnum* parameter as a block number.

(0:7) These bits are reserved for system use.

The bits in the *aoptions* parameter are set in the same manner as those in the *foptions* parameter.

The FOPEN intrinsic can return the following condition codes:

| | |
|-----|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CCE | Request granted; the file was opened. |
| CCG | (Not returned by this intrinsic.) |
| CCL | Request denied. Another process already has exclusive or semi-exclusive access to the file, or an odd number of bytes was specified in the <i>inhibit-buffering</i> option. <u>When the condition code CCL occurs, a file number of 0 is returned to the user's process.</u> |

EXAMPLE:

To open a particular file, the user issues the following intrinsic call.

```
FILEA := FOPEN (F1,%41,,REC,DV,,,,7000D);
```

The *filenumber* is returned to the word *FILEA*, and the condition code *CCE* results. The file characteristics are

| | |
|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>formaldesignator</i> | = <i>FX22</i> (contained in the byte-array <i>F1</i>) |
| <i>foptions</i> | = Domain: system domain ASCII/Binary: Binary Default designator: \$STDIN Record format: Fixed-length Carriage control: None Disallow :FILE: Not active |

| | |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| aoptions | = <i>(Default Values)</i> <i>Access type: Read access only</i> <i>Multirecord: Non-multirecord mode</i> <i>Dynamic locking: Disallowed</i> <i>Exclusive: Share-access specified</i> <i>Inhibit-buffering: Not selected</i> |
| recsize | = <i>As specified by the value of REC.</i> |
| device | = <i>DISKA, as specified in the byte-array DV</i> |
| formmsg | = <i>None (default)</i> |
| userlabels | = <i>None (default)</i> |
| blockfactor | = <i>128/recsize (default)</i> |
| numbuffers | = <i>2 (default)</i> |
| filesize | = <i>7000 logical records</i> |
| numextents | = <i>System default (8)</i> |
| initialloc | = <i>1 (default)</i> |
| filecode | = <i>0 (default)</i> |

CLOSING FILES

To terminate access to a file, the user issues the FCLOSE intrinsic call. (This intrinsic applies to files on all devices.) This intrinsic deletes the buffers and control blocks through which the user's process accessed the file. It also deallocates the device on which the file resides. Additionally, it may change the disposition of the file. If the programmer does not issue FCLOSE calls for all files opened by his process, such calls are automatically issued by MPE/3000 when the process terminates. (When a file on tape is saved, the tape is rewound. New, saved tape files are not loaded, and remain off-line.)

The FCLOSE intrinsic call is written in this format:

```
PROCEDURE FCLOSE (filenum, disposition, seccode);
```

```
VALUE filenum, disposition, seccode;
```

```
INTEGER filenum, disposition, seccode;
```

```
OPTION, EXTERNAL;
```

The FCLOSE call parameters are shown below:

- filenum* A word identifier supplying the file number of the file to be closed, through SPL/3000 conventions. (*FROM FOPEN*)
- disposition* An integer indicating the disposition of the file, significant only for files on disc and magnetic tape. (This disposition can be overridden by a corresponding parameter in a :FILE command entered prior to program execution.) The disposition options are defined by two bit-fields, as follows:

| Bit Nos. | Option |
|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| (13:3) | <i>Domain Disposition</i> |
| | 0 = <i>No change</i> . The disposition code remains as it was before the file was opened. Thus, if the file is new, it is deleted by FCLOSE; otherwise, the file is assigned to the domain to which it previously belonged. |
| | 1 = <i>Permanent File</i> . The file is saved in the system domain. If the file is a new or old temporary file on disc, an entry is created for it in the system file directory. (An error code is returned if a file of the same name already exists in the directory.) If it is an old permanent file on disc, this disposition value has no effect. If the file is stored on magnetic tape, that tape is rewound and unloaded. |
| | 2 = <i>Temporary Job File (Rewound)</i> . The file is retained in the user's temporary (job/session) file domain and can thus be reopened by any process within the job/session. The uniqueness of the file name is checked; if a file of this name already exists, an error code is returned. If the file resides on magnetic tape, the tape is rewound but not unloaded. |
| | 3 = <i>Temporary Job File (Not Rewound)</i> . This option has the same effect as Disposition Code 2, except that tape files are <i>not</i> rewind. |
| | 4 = <i>Released File</i> . The file is deleted from the system. |

The default value for this field is code 0 (no change).

| | |
|--------|-------------------------------------------------------------------------------------------------------------------|
| (12:1) | <i>Disc Space Disposition</i> |
| | <u>1</u> = Returns to the system any disc space allocated beyond the end-of-file indicator. (<i>IBM "RLSE"</i>) |
| | <u>0</u> = Does not return any disc space allocated beyond the end-of-file indicator. |

The default value for this field is code 0 (no return).

If more than one access is in effect for the file, its disposition is not affected until the last access terminates (with an FCLOSE call). Then, the effective disposition is the smallest non-zero disposition parameter specified among all FCLOSE calls issued against the file.

seccode An integer denoting the type of security initially applied to the file, significant only for new permanent files. The options are:

- 0 Unrestricted access--the file can be accessed by any user, unless prohibited by current MPE/3000 security provisions.
- 1 Private file-creator security--the file can be accessed only by its creator.

The default *seccode* value is 0.

The following condition codes can be returned:

- CCE The file was closed successfully.
- CCG (This code is not returned.)
- CCL The file was not closed, perhaps because an incorrect *filenum* parameter was specified; or because another file with the same name and disposition exists in the system.

When a process is ended within a job, FCLOSE (0,0) is issued against all open files in the job temporary file domain. When a job is terminated, all job temporary files are purged.

EXAMPLE

The following intrinsic call closes a file whose *filenumber* is supplied through the identifier *FILEX*. The file is to be saved as a permanent file (Disposition Code 1), having private creator file security.

```
FCLOSE (FILEX,1,1)
```

The CCE condition code is returned to the user's process.

READING SEQUENTIAL FILES

To read a logical record (or a portion of such a record) from a sequential file (on any device) to the user's data stack, the user issues the FREAD intrinsic call. The format of this call is

```
INTEGER PROCEDURE FREAD (filenum,target,tcount) ;
```

```
VALUE      filenum,tcount;
```

```
INTEGER    filenum,tcount;
```

```
ARRAY     target;
```

```
OPTION EXTERNAL;
```

The FREAD intrinsic returns (as the value of FREAD) an integer showing the length of the information read.

The FREAD call parameters are

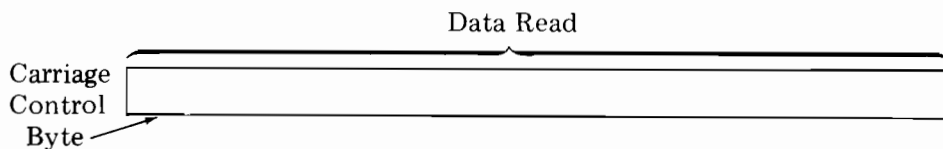
- filenum* A word identifier supplying the filenumber of the file to be read, through SPL/3000 conventions.
- target* The array to which the record is to be transferred.
- tcount* An integer specifying the number of words or bytes to be transferred. If this value is positive, it signifies the length in *words*; if it is negative, it signifies the length in *bytes*; if it is zero, no transfer occurs. If *tcount* is less than the size of the record, only the first *tcount* bytes or words are read from the record.

If *tcount* is larger than the size of the physical record (i.e., block), and the *multirecord* option was *not* specified in FOPEN, transfer is limited to the length of the physical record. (But if the *multirecord* option *was* specified, the remaining words in *tcount* will be read from the next successive records.)

When a record is read, the FREAD intrinsic returns to the user's program a positive value showing the length of the information transferred. If the *tcount* parameter in the FREAD call was positive, the value returned represents a *word* count; if the *tcount* parameter was negative, the value returned is a *byte* count. The logical record pointer is now set at the beginning of the next logical record in the file.

When the logical end-of-data is encountered during reading, the CCG condition code is returned to the user's process. On magnetic tape, the end-of-data can be denoted by a physical indicator such as a tape mark; on disc, it occurs when the last logical record written to the file is passed. If the file is imbedded in an input source containing MPE/3000 commands, the end-of-data is indicated when an :EOD command is encountered (but that command itself is not returned to the user). (The end-of-data is indicated, on \$STDIN by any MPE/3000 command; on \$STDINX, it is indicated by :JOB, :EOJ, :EOD, and :DATA.)

When an old file containing carriage-control characters (supplied through the *control* parameter of the FWRITE intrinsic) is read, and the carriage-control *foption* (FOPEN intrinsic) or CCTL parameter (:FILE command) is specified, the carriage-control byte is read:



(If file has carriage control specified)

The condition codes possible are

| | |
|-----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CCE | The information was read. |
| CCG | The logical end-of-data was encountered during reading. |
| CCL | The information was not read because an error occurred; a terminal read was terminated by a special character (as specified in the FCONTROL intrinsic); or a tape error was recovered and the FSETMODE option was enabled. |

EXAMPLES:

To read a 20-word logical record from a sequential file whose filenum is supplied through the identifier F1 to the array R1 in the stack, the user enters the following intrinsic call. The length of the information to be transferred is 10 words. The length of the information actually transferred is stored in the word COUNT.

COUNT := FREAD (F1,R1,20);

To read the first ten words from the next record in this file to the array R2, the user enters the following call. The length of the record transferred is stored in the word COUNTX.

COUNTX := FREAD (F1,R2,10);

READING DIRECT-ACCESS FILES

To read a logical record (or a portion of such a record) from a direct-access disc file to the user's data stack, the user enters the FREADDIR intrinsic call. This call may only be issued for a disc file composed of fixed-length or undefined-length records. The format of FREADDIR is

```
PROCEDURE FREADDIR (filenum,target,tcount,recnum) ;  
  
VALUE filenum,tcount,recnum;  
  
INTEGER filenum;  
  
ARRAY target;  
  
INTEGER tcount;  
  
DOUBLE recnum;  
  
OPTION EXTERNAL;
```

The FREADDIR call parameters are

| | |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>filename</i> | A word identifier supplying the filename of the file to be read, through SPL/3000 conventions. |
| <i>target</i> | An array to which the logical record is to be transferred. |
| <i>tcount</i> | An integer specifying the number of words or bytes to be transferred. If this value is positive, it signifies <i>words</i> ; if negative, it denotes bytes; if zero, no transfer occurs. If tcount is less than the size of the record, only the first tcount bytes or words are read from the record. If tcount is larger than the actual size of the logical record and the <i>multirecord</i> option was <i>not</i> specified in FOPEN, the transfer is limited to the length of the logical record; otherwise, the excess words or bytes will be read from the next successive records. |
| <i>recnum</i> | A double-word integer indicating the relative number, in the file, of the logical record to be read; the first record is indicated by 0D. |

After the FREADDIR intrinsic is executed, the logical record pointer is now set at the beginning of the next logical record (or first logical record of the next block for NOBUF files) in the file.

As disc extents are initially allocated to a direct-access file, they are filled with binary zeros (for binary files) or ASCII blanks (for ASCII files). Thus, if a record is read that has never been written upon, binary zeros or ASCII blanks are transmitted to the user's stack.

The condition codes possible are

| | |
|-----|---------------------------------------------------------|
| CCE | The information specified was read. |
| CCG | The logical end-of-data was encountered during reading. |
| CCL | The information was not read because an error occurred. |

EXAMPLES:

To read the first logical record (containing 60 words) from a direct-access file whose filename is supplied through the identifier FNUM, to a stack array named R19, the user enters the following. (In SPL/3000, a double-word integer is always indicated by the suffix D.)

```
FREADDIR (FNUM,R19,60,0D);
```

To read the first 20 bytes from logical record 24 in the same file, the user enters:

```
FREADDIR (FNUM,R19,-20,24D);
```

OPTIMIZING DIRECT-ACCESS FILE READING

The user can enhance the direct-access of disc files by issuing the **FREADSEEK** intrinsic call. This call is used when the need for a certain record is known before its transfer to the user's stack, by a **FREADDIR** call, is actually required. **FREADSEEK** directs MPE/3000 to move the record from disc into a buffer in anticipation of the **FREADDIR** call, which subsequently moves the record directly to the stack.

NOTE: The FREADSEEK intrinsic call can only be issued against a file for which input/output buffering and fixed/undefined length record formats are in effect.

The format of the FREADSEEK call is

PROCEDURE **FREADSEEK (filename,recnum)**

VALUE filename,recnum;

INTEGER filename;

DOUBLE recnum;

OPTION EXTERNAL;

The **FREADSEEK** parameters are

filename A word supplying the filename of the file to be read, through SPL/3000 conventions.

recnum A double-word integer indicating the relative number of the logical record to be read; the first record is indicated by 0D.

The condition codes possible are

CCE The request was granted.

CCG A logical end-of-file indication was encountered.

CCL The request was not granted because of error.

EXAMPLE:

To perform pre-read buffering on the third record of the file whose filename is supplied through the identifier ISAAC, the user enters:

```
FREADSEEK (ISAAC,3D);  
.  
.  
.  
FREADDIR (ISAAC,STAK,60,3D);
```

WRITING ON SEQUENTIAL FILES

To write a logical record (or a portion of such a record) from the user's stack to a sequential file, (on any device) the user issues the FWRITE intrinsic call:

```
PROCEDURE FWRITE (filenum, target, tcount, control) ;
```

```
VALUE filenum, tcount, control;
```

```
INTEGER filenum, tcount;
```

```
ARRAY target;
```

```
LOGICAL control;
```

```
OPTION EXTERNAL;
```

The FWRITE call parameters are

filenum A word identifier supplying the filenumber of the file to be written on, through SPL/3000 conventions.

target An array in the stack that contains the record to be written.

tcount An integer specifying the number of words or bytes to be written from the record. If this value is positive, it signifies *words*; if it is negative, it signifies *bytes*; if zero, no transfer occurs. If *tcount* is less than the *recsize* parameter associated with the record, only the first *tcount* words or bytes are written.

If *tcount* is larger than the *recsize* value, and the *multirecord* aoption was *not* specified in FOPEN, the FWRITE request is refused and condition code CCL is returned; otherwise, the excess words or bytes are written to the next successive records. (For files for which carriage control is specified, the actual data transferred is limited to *recsize* minus one byte.)

control A logical value representing a carriage control code, effective if the file is transferred to a line printer or terminal. This parameter is effective only for files opened with carriage control specified. The options are

0 = To print the full record transferred, using single spacing. This results in a maximum of 132 characters per printed line.

1 = To use the first character of the data written to signify space control, and suppress this character on the printed output. This results in a maximum of 132 characters of data per printed line. (Permissible control characters are shown in Figure 6-5.)

Any other octal code from Figure 6-5. = To use this code to determine space control, and print the full record transferred. This results in a maximum of 132 characters per printed line. (No data will be transferred if the octal code is 100 through 103 or 400 through 403, or if the octal code is 1 and the embedded control is octal 100 through 103.)

If the *control* parameter is not 1, and *tcount* is 0, only the space control is executed--no data is transferred.

The effect of the FWRITE *control* parameter in combination with the FOPEN carriage control *foption* (or overriding :FILE command CCTL/ NOCCTL parameter) upon the data written is summarized in Figure 6-6.

The user determines whether the carriage-control directive takes effect before printing (pre-space movement) or after printing (post-space movement), through the FCONTROL intrinsic, described later in this section.

All of the carriage control codes listed in Figure 6-5 may be used as the value of the *param* parameter to FCONTROL (when *controlcode* = 1), regardless of whether the file is opened with CCTL or NOCCTL. When the file is opened with CCTL, these carriage control codes may be used in either of the following ways via FWRITE:

- a. as the value of the *control* parameter;
- b. when *control* = 1, as the first byte of the *target* array.

The default carriage control codes are post spacing with automatic page eject.

When information is written to a fixed-length record (and *nobuff* was not specified in FOPEN), any unused portion of the record will be padded with binary zeros (for a binary file) or ASCII blanks (for an ASCII file).

When the FWRITE intrinsic is executed, the logical record pointer is set to the record immediately following the record just written (or the first logical record in the next block for NOBUF files).

When an FWRITE call writes a record beyond the current logical end-of-file indicator, this indicator is advanced to a further location.

When the physical bounds of the file prevent further writing (because all allowable extents are filled), the end-of-file condition code (CCG) is returned to the user's process. This intrinsic cannot access user labels.

The condition codes possible are

- | | |
|-----|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CCE | The request was granted. |
| CCG | The physical bounds of the file prevented further writing; all disc extents were filled. |
| CCL | The request was not granted because an error occurred (such as <i>tcount</i> exceeding the size of the record in non-multirecord mode); or the FSETMODE option is enabled to signify recovered tape errors; or the end-of-tape marker was sensed. |

| Octal Code | ASCII Symbol | Carriage Action |
|------------------------------------------------------------------------------------|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| %40 | " " | *Single-space. |
| %60 | " 0 " | *Double-space. |
| %61 | " 1 " | Page-eject (form-feed). |
| %53 | " + " | No space, return (next printing at column 1). |
| %2nn | | Space nn lines. (No automatic page eject.) |
| (where n is any digit from 0 through 7) | | |
| %300 | | Page-eject (Tape Channel 1). |
| %301 | | Skip to bottom of form (Tape Channel 2). |
| %302 | | Single-spacing (with automatic page eject). (Tape Channel 3.) |
| %303 | | Single-space on next odd-numbered line (with automatic page eject). (Tape Channel 4.) |
| %304 | | Triple-space (with automatic page eject). (Tape Channel 5.) |
| %305 | | Space 1/2 page (with automatic page eject). (Tape Channel 6.) |
| %306 | | Space 1/4 page (with automatic page eject). (Tape Channel 7.) |
| %307 | | Space 1/6 page (with automatic page eject). (Tape Channel 8.) |
| %310 | | Space to bottom of form. (Tape Channel 9.) |
| %311 | | Skip to Tape Channel 10. (User option.) |
| %312 | | Skip to Tape Channel 11. (User option.) |
| %313 | | Skip to Tape Channel 12. (User option.) |
| %320 | | No space, no return. (Next printing physically follows this.) |
| %0 – %37 | } | **Same as %40 |
| %41 – %52 | | |
| %54 – %57 | | |
| %62 – %77 | | |
| %314 – %317 | | |
| %321 – %377 | | |
| %400 or %100 | | Set <u>post-space</u> movement option; this first prints, then spaces. If previous option set was pre-space movement option, the driver outputs a line (and suppresses spacing) to clear the buffer. |
| %401 or %101 | | Set <u>pre-space</u> movement option; this first spaces, then prints. |
| %402 or %102 | | Set single-space option, with automatic page eject (60 lines per page). |
| %403 or %103 | | Set single-space option <i>without</i> automatic page eject (66 lines per page). |
| * Spacing with or without automatic page eject can be selected. | | |
| ** Future MPE/3000 requirements may necessitate redefinition of these octal codes. | | |

DEFAULT

Figure 6-5. Carriage-Control Directives

| | | FWRITE Control Parameter | | |
|--------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| | | = 0 | = 1 | = Greater than 1 |
| FOPEN or :FILE | | | | |
| Carriage- Control Option Specified or CCTL | <p>Byte 1</p> <p>∅ record</p> <p>Data output contains 132 characters; the first byte contains 0.</p> | <p>record</p> <p>Data output contains 132 characters; the carriage control character in the first byte is suppressed if output is to a list device.</p> | <p>Byte 1</p> <p>control record</p> <p>Data output contains 132 characters; the first character is a carriage-control character specified by the FWRITE control parameter.</p> | |
| Carriage Control Option <i>not</i> specified or NOCCTL | <p>record</p> <p>Data output contains 132 characters.</p> | <p>record</p> <p>Data output contains 132 characters.</p> | <p>record</p> <p>Data output contains 132 characters.</p> | |

EFFECT ON DATA OUTPUT

Figure 6-6. Carriage-Control Summary

EXAMPLE:

To write a 20-word logical record from the user's array *INFO* to the file designated by the identifier *OUTGO*, the user issues the following intrinsic call. When the record is printed, single spacing is used.

```
FWRITE (OUTGO,INFO,20,0);
```

To write the six bytes from the next record in the array *NEWINFO*, to the file designated by *OUTGO2*, the user writes the following. (The first character in the record is used as a control character.)

```
FWRITE (OUTGO2,NEWINFO,-6,1);
```

WRITING ON DIRECT-ACCESS FILES

To write a logical record (or a portion of such a record) from the user's stack to a direct-access disc file, the user issues the **FWRITEDIR** intrinsic call. This call may only be issued for a disc file composed of fixed or undefined length records. Its format is

PROCEDURE

```
FWRITEDIR (filename,target,tcount,recnum) ;
```

VALUE filename,tcount,recnum;

INTEGER filename,tcount;

ARRAY target;

INTEGER tcount;

DOUBLE recnum;

OPTION EXTERNAL;

The **FWRITEDIR** call parameters are

- | | |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>filename</i> | A word identifier supplying the filename of the file to be written on, through SPL/3000 conventions. |
| <i>target</i> | An array in the stack that contains the record to be written. |
| <i>tcount</i> | An integer specifying the number of words or bytes to be written from the record. If this value is positive, it signifies <i>words</i> ; if it is negative, it signifies <i>bytes</i> ; if zero, no transfer occurs. If <i>tcount</i> is less than the <i>recsize</i> parameter associated with the record, only the first <i>tcount</i> words or bytes are written. If <i>tcount</i> is larger than the <i>recsize</i> value and the <i>multirecord</i> option was not specified in FOPEN , the FWRITEDIR request is refused and condition code CCL is returned; otherwise, the excess words or bytes are written to the next successive records. |

recnum A double-integer indicating the relative number of the logical record (or block number for NOBUF files) to be written; the first record is indicated by 0D.

When information is written to a fixed-length record (and *nobuff* was not specified in FOPEN), any unused portion of the record will be padded with binary zeros (for a binary file) or ASCII blanks (for an ASCII file).

When the FWRITEDIR intrinsic is executed, the logical record pointer is set to the record immediately following the record just written (or the first logical record of the next block for NOBUF files).

When a FWRITEDIR call writes a record beyond the current logical end-of-file indicator, this indicator is advanced to a further location. This can result in creation of dummy records to pad the records between the previous end-of-file and the newly-written record. These dummy records are filled with binary zeros (on a binary file) or with ASCII blanks (on an ASCII file).

When the physical bounds of the file prevent further writing (because all allowable extents are filled), the end-of-file condition code (CCG) is returned to the user's program.

The condition codes possible are

| | |
|-----|----------------------------------------------------------|
| CCE | The request was granted. |
| CCG | The physical end-of-file was encountered during writing. |
| CCL | The request was not granted because an error occurred. |

EXAMPLES:

To write logical record number 15 (containing 70-words) from the array DATA to the file identified by OUTX, the user writes:

```
FWRITEDIR (OUTX,DATA,70,15D);
```

To write three words from the array NSTACK to record 16 in the file identified by OUTDATA, the user writes:

```
FWRITEDIR (OUTDATA,NSTACK,3,16D);
```

READING LABELS

To read a user-defined label from a disc file, the user calls the FREADLABEL intrinsic. Before reading occurs, the user's *read-access* capability is verified. (Note that MPE/3000 automatically skips over any unread user labels when the first FREAD call is issued against a file.)

```
PROCEDURE FREADLABEL (filenum, target, tcount, labelid);  
VALUE filenum, tcount, labelid;  
INTEGER filenum, tcount, labelid;  
ARRAY target;  
OPTION VARIABLE, EXTERNAL;
```



The FREADLABEL parameters are as follows: (Note that *tcount* and *labelid* are optional.)

| | |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>filenum</i> | A word identifier supplying the filenumber of the file whose label is to be read, through SPL/3000 conventions. |
| <i>target</i> | An array (in the stack) to which the label is to be transferred. |
| <i>tcount</i> | An integer specifying the number of words to be transferred from the label. For disc files, <i>tcount</i> is limited to 128 words. If <i>tcount</i> is omitted, a default value of 128 words is assigned. |
| <i>labelid</i> | An integer specifying the label number required. If <i>labelid</i> is omitted, a default value of 0 is assigned. |

The possible condition codes are:

| | |
|-----|-----------------------------------------------------------------|
| CCE | The label was read. |
| CCG | The intrinsic referenced a label beyond the last label written. |
| CCL | The label was not read because an error occurred. |

WRITING LABELS

To write a user-defined label onto a disc file, the user calls the `FWRITELABEL` intrinsic.

```
PROCEDURE FWRITELABEL (filenum, target, tcount, labelid);  
VALUE filenum, tcount, labelid;  
INTEGER filenum, tcount, labelid;  
ARRAY target;  
OPTION VARIABLE, EXTERNAL;
```

The `FWRITELABEL` parameters follow. (Note that `tcount` and `labelid` are optional.)

| | |
|----------------|------------------------------------------------------------------------------------------------------------------------------|
| <i>filenum</i> | A word identifier supplying the filenumber of the file to which the label is to be written, through SPL/3000 conventions. |
| <i>target</i> | An array in the stack from which the label is to be transferred. |
| <i>tcount</i> | An integer specifying, for disc files, the number of words to be transferred from the array. The default value is 128 words. |
| <i>labelid</i> | An integer specifying the number of the label required. If omitted, a default value of 0 is assigned. |

The condition codes possible are:

| | |
|-----|------------------------------------------------------------------------------------------------------------------------------|
| CCE | The label was written. |
| CCG | The calling process attempted to write a label beyond the limit specified (in <code>FOPEN</code>) when the file was opened. |
| CCL | The label was not written because an error occurred. |

UPDATING FILES

To update a logical record residing on a direct-access device (disc), the user issues the FUPDATE intrinsic call. This intrinsic only applies to fixed and undefined length record files. It affects the logical record (or block for NOBUF files) last referenced by any intrinsic call for the file named; it moves the specified information from the user's stack into this record. The file containing this record must have been opened with the *update* option specified in the FOPEN call. The FUPDATE call format is

```
PROCEDURE FUPDATE (filenum,target,tcount) ;
```

```
VALUE filenum,tcount;
```

```
INTEGER filenum,tcount;
```

```
ARRAY target;
```

```
OPTION EXTERNAL;
```

The FUPDATE call parameters are

- | | |
|----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>filenum</i> | A word identifier supplying the filenumber of the file to be updated, through SPL/3000 conventions. |
| <i>target</i> | An array in the stack that contains the record to be written in the updating. |
| <i>tcount</i> | An integer specifying the number of words or bytes to be written from the record. If this value is positive, it signifies <i>words</i> ; if it is negative, it signifies bytes; if zero, no transfer occurs. If <i>tcount</i> is less than the <i>reclsize</i> parameter associated with the record, only the first <i>tcount</i> words or bytes are written. For buffered files, <i>tcount</i> is limited to <i>reclsize</i> . For NOBUF files, <i>tcount</i> is limited to the block size. FUPDATE cannot perform multirecord updates. |

The condition codes possible are

- | | |
|-----|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CCE | The request was granted. |
| CCG | An end-of-file condition was encountered during updating. |
| CCL | The request was not granted because of an error, such as the file not residing on a direct-access device, or <i>tcount</i> exceeding the size of the record when multirecord mode is not in effect. |

EXAMPLE:

To update the last record referenced in the file identified by *VNAME*, the user wants to write a record from the array *SAREA* (in his stack) onto this record. He can do so by entering:

FUPDATE (VNAME,SAREA,50);

SPACING ON SEQUENTIAL FILES

The user can space forward or backward on a sequential disc or tape file by using the **FSPACE** intrinsic call. (This results in re-setting the logical record pointer.) The **FSPACE** call format is

PROCEDURE *FSPACE (filenum,displacement) ;*

VALUE *filenum, displacement;*

INTEGER *filenum,displacement;*

OPTION EXTERNAL;

The **FSPACE** call parameters are

| | |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>filenum</i> | A word identifier supplying the filenumber of the file on which spacing is to be done, through SPL/3000 conventions. |
| <i>displacement</i> | A signed integer indicating the number of logical records for buffered disc files (or blocks for NOBUF disc files and all tape files) to be spaced over, relative to the current position of the file. A positive value signifies forward spacing; a negative value requests backward spacing. (For positive values, the sign is optional.) |

The condition codes possible are

| | |
|------------|-------------------------------------------------------------------------------------------------------------|
| CCE | The request was granted. |
| CCG | A logical end-of-file indicator was encountered during spacing. |
| CCL | The request was not granted because an error occurred; the file resides on a device that prohibits spacing. |

EXAMPLE:

To space forward 16 logical records on the file identified by *F20*, the user issues:

FSPACE (F20,16);

RESETTING LOGICAL RECORD POINTER

The programmer can reset the logical record pointer for a sequential disc file, containing only fixed-length records, at any logical record in the file. When the next FREAD or FWRITE request is issued for the file, this record will be the one read or written. Positioning is done through the FPOINT intrinsic call. (This intrinsic applies to fixed and undefined length record files.)

PROCEDURE *FPOINT (filenum,recnum) ;*

VALUE filenum,recnum;

INTEGER filenum;

DOUBLE recnum;

OPTION EXTERNAL;

The FPOINT call parameters are

filenum A word identifier supplying the filenumber of the file on which spacing is to be done, through the conventions of SPL/3000.

recnum A positive double integer representing the relative logical record (or block number for NOBUF files) at which the file is to be positioned. (The number of the first record is zero.)

The condition codes are

CCE The request was granted.

CCG The user requested positioning at a point beyond the physical end-of-file.

CCL The request was not granted because an error occurred.

EXAMPLE:

To position a file identified by FILE2 at the 40th logical record within it, the user enters:

FPOINT (FILE2,39D);

OBTAINING FILE ACCESS INFORMATION

Once the user opens a file on any device, he can request access and status information about that file at any point in his program.

The parameters supplied in the FOPEN call, or corresponding information from any :FILE command or file label that overrides these parameters, are included in the information returned. The intrinsic call format is

PROCEDURE

| |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>FGETINFO</i> (<i>filenum, filename, foptions, aoptions, recsize, devtype, ldnum, hdaddr, filecode, recpt, eof, flimit, logcount, physcount, blksize, extsize, numextents, userlabels, creatorid, labaddr</i>); |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

VALUE *filenum*;

INTEGER *filenum, recsize, devtype, filecode, blksize, numextents, userlabels*;

BYTE ARRAY *filename, creatorid*;

LOGICAL *foptions, aoptions, ldnum, hdaddr, extsize*;

DOUBLE *recptr, eof, flimit, logcount, physcount, labaddr*;

OPTION VARIABLE, EXTERNAL;

The FGETINFO call parameters are

filenum A word identifier supplying the filenumber of the file about which information is requested, through SPL/3000 conventions.

filename A byte-array to which is returned the actual designator of the file being referenced, in this format:

 f.g.a

 f = The local file name.

 g = The group name (supplied or implicit).

 a = The account name (supplied or implicit).

The byte-array must be 28-bytes long. When the actual designator is returned, unused bytes in the array are blank-filled to the right. A nameless file will return an empty string.

foptions A word to which is returned a bit string representing the file options currently effective. The format is the same as that of the *foptions* parameter of FOPEN.

aoptions A word to which is returned a bit string representing the access options which are currently effective. The format is the same as that of the aoptions parameter of FOPEN.

recsize A word to which is returned the logical record size associated with the file. If the file was created as a binary type, this value is positive and expresses the size in *words*. If the file was created as an ASCII type, this value is negative and expresses the size in *bytes*.

devtype A word to which is returned the type and subtype of device being used for the file, where

(0:8) = device subtype

(8:8) = device type

(Refer to Appendix A of MPE/3000 Console Operator's Guide for device types and subtypes.)

If the file is not spooled, which can be determined from **hdaddr** (0:8), the returned **devtype** is actual. The same is true if the file is spooled and was opened via logical device number. However, if an output file is spooled and was opened by device class name, **devtype** contains the type and subtype of the first device in the class, which may be different from the device actually used.

ldnum A word to which is returned the logical device number associated with the device on which the file resides.

If the file is spooled, then **ldnum** will be a virtual device number which does not correspond to the system configuration I/O device list. Although virtual device numbers may be used programmatically as real logical device numbers are used for successive FOPENs, it cannot be the target of user or console commands.

hdaddr A word to which is returned the hardware address of the device, where:

Bits (0:8) = DRT number

Bits (8:8) = Unit number

If the device is spooled, the DRT number will be zero and the unit number is undefined.

filecode A word to which is returned the value recorded with the file as its *file code* (for disc files).

recptr A double word to which is returned a double integer representing the current logical record pointer setting. This is the displacement in logical records from Record No. 0 in the file. It identifies the record that would next be accessed by an FREAD or FWRITE call.

| | |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| eof | A double word to which is returned a double positive integer equal to the number of logical records currently in the file. (If the file does not reside on disc, this value will be zero.) |
| flimit | A double word to which is returned a double positive integer representing the number of the last logical record that could ever exist in the file, because of the physical limits of the file. (If the file does not reside on disc, this value will be zero.) |
| logcount | A double word to which is returned a double positive integer representing the total number of logical records passed to and from the user during the current access of the file. |
| physcount | A double word to which is returned a double positive integer representing the total number of physical input/output operations performed within this process against the file since the last FOPEN call. |
| blksize | A word to which is returned the block size associated with the file. If the file was created as a binary type, this value is positive and expresses the size in words. If the file was created as an ASCII type, this value is negative and shows the size in bytes. |
| extsize | A word to which is returned the disc extent size associated with the file (in sectors). |
| numextent | A word to which is returned the maximum number of disc extents allowable for the file. |
| userlabels | A word to which is returned the number of user header labels defined for the file when it was created. If the file is not a disc file, this number is 0. (When an old file is opened for overwrite-output, the value of <i>userlabels</i> is not reset; old user labels are not destroyed.) |
| creatorid | A byte-array to which is returned the eight-byte name of the user who created the file. If the file is not a disc file, blanks are returned. |
| labaddr | A double word to which is returned the sector address of the label of the file. If the file is not a disc file, this value is 0. The high-order eight bits show the logical device number; the next 24 bits show the absolute disc address. |

The condition codes are

| | |
|------------|--------------------------------------------------------|
| CCE | The request was granted. |
| CCG | (This condition code is not returned.) |
| CCL | The request was not granted because an error occurred. |

EXAMPLE:

To request file access and status information about a file identified as F5, the user can enter the following call. This will return the file designator to the word FNDATA, the logical record size to the word FRDATA, and the type of device on which the file resides to FDDATA.

FGETINFO (FS,FNDATA,,,FRDATA,FDDATA);

After the intrinsic is executed, the values returned are as follows:

| Byte-Array or Word | Content | Interpretation |
|-----------------------|----------------------------------------|--------------------------------------------------------------------------------|
| FNDATA | FILESAM.GP1 | The file named FILESAM in the group GP1 in the user's log-on account. |
| FRDATA | 100 (Decimal equivalent of content) | The file is a binary file containing fixed-length records each 100 words long. |
| FDDATA | 24 (Decimal equivalent of content) | The file resides on magnetic tape. |

OBTAINING FILE-ERROR INFORMATION

When a file intrinsic returns a condition code indicating a physical input/output error, the programmer may require more details about the error in order to correct it. He can request this information with the FCHECK intrinsic call. (This intrinsic applies to files on any device.)

FCHECK is the only intrinsic that accepts zero as a legal *filenum* parameter value. When zero is specified, the returned error code reflects the status of the last call to FOPEN; zero is necessary to allow one to reference an erroneously-opened file. The format of FCHECK is

PROCEDURE *FCHECK (filenum,errorcode,tlog,blknum,numrec) ;*

VALUE filenum;

INTEGER filenum,errorcode,tlog,numrec;

DOUBLE blknum;

OPTION VARIABLE, EXTERNAL;

The FCHECK call parameters are

| | |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>filenum</i> | A word identifier supplying the file number of the file for which error information is to be returned, through SPL/3000 conventions. |
| <i>errorcode</i> | A word to which is returned a 16-bit code, explained below, specifying the type of error that occurred. If the previous operation was successful, all bits are set to zero. |
| <i>tlog</i> | A word to which is returned the transmission-log value recorded when an erroneous data transfer occurs. This specifies the number of words not read or written (those left over) as the result of the input/output error. |
| <i>blknum</i> | A double word to which is returned the relative number of the block involved in the error. |
| <i>numrec</i> | A word to which is returned the number of logical records in the bad block. |

In the 16 bits returned to the word specified by the *errorcode* parameter, the low-order eight bits contain the error-type code that shows what kind of error occurred. The high-order eight bits are set to zero.

The following codes are returned by FCHECK:

| Code (Decimal) | Meaning |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 20 | Invalid operation. |
| 21 | Data parity error. |
| 22 | Software time-out. |
| 23 | End of tape. |
| 24 | Unit not ready. |
| 25 | No write-ring on tape. |
| 26 | Transmission error. |
| 27 | Input/output time-out. |
| 28 | Timing error or data overrun. |
| 29 | Start input/output (SIO) failure. |
| 30 | Unit failure. |
| 31 | End-of-input record (special character terminator). |
| 32 | Software abort. |
| 33 | Data lost. |
| 34 | Unit not on-line. |
| 35 | Data set not ready. |
| 36 | Invalid disc address. |
| 37 | Invalid memory address. |
| 38 | Tape parity error. |
| 39 | Recovered tape error. |
| 40 | Operation inconsistent with access-type. |
| 41 | Operation inconsistent with record type. |
| 42 | Operation inconsistent with device type. |
| 43 | The <i>tcount</i> parameter value exceeded the <i>recsize</i> parameter value in this intrinsic, but the <i>multirecord access</i> option was not specified in the currently-effective FOPEN intrinsic. |
| 44 | The FUPDATE intrinsic was called, but the file was positioned at record zero. (FUPDATE must reference the last record read, but no previous record was, in fact, read.) |
| 45 | Privileged-file violation. |
| 46 | Insufficient disc space. |
| 47 | Input/output error occurs on a file label. |
| 48 | Invalid operation due to multiple file access. |
| 49 | Unimplemented function. |
| 50 | The account referenced does not exist. |
| 51 | The group referenced does not exist. |
| 52 | The file referenced does not exist in the system file domain. |
| 53 | The file referenced does not exist in the job temporary file domain. |

| Code (Decimal) | Meaning |
|-------------------|-------------------------------------------------------------------------------------------------------------------|
| 54 | The file reference is invalid. |
| 55 | The device referenced is not available. |
| 56 | The device specification is invalid. |
| 57 | Virtual memory is not sufficient for the file specified. |
| 58 | The file was not passed. |
| 59 | Standard label violation. |
| 60 | Global RIN not available. |
| 61 | Group disc file space exceeded. |
| 62 | Account disc file space exceeded. |
| 63 | Non-sharable device capability not assigned. |
| 64 | Multiple RIN capability required. |
| 71 | Too many files for process. |
| 72 | Invalid file number. |
| 73 | Bounds-check violation. |
| 80 | Configured maximum number of spool sectors would be exceeded by this output request. |
| 81 | No SPOOL class defined in system. |
| 82 | Insufficient space in SPOOL class to honor this output request. |
| 83 | I/O error while accessing next spool extent. |
| 84 | The next extent in this spool resides on a device which is unavailable to the system (i.e., the device is =DOWN). |
| 85 | Operation inconsistent with spooling; e.g., attempt to read hardware status. |
| 86 | An attempt has been made to access a non-existent spool. |
| 87 | Bad spool block. |
| 89 | Power failure. |
| 90 | The calling process requested exclusive access to a file to which another process has access. |
| 91 | The calling process requested access to a file to which another process has exclusive access. |
| 92 | Lockword violation. |
| 93 | Security violation. |
| 94 | Creator conflict in use of FRENAME intrinsic. |
| 100 | Duplicate file name in the system file directory. |
| 101 | Duplicate file name in the job temporary file directory. |
| 102 | Directory input/output error. |
| 103 | System directory overflow. |
| 104 | Job temporary directory overflow. |

| | |
|-----|------------------------------------------------------------------------------------|
| 106 | Extent size exceeds maximum. |
| 107 | Insufficient space for number of user labels specified. |
| 110 | The intrinsic attempted to save a system file in the job temporary file directory. |
| 111 | The user lacks Save file (SF) capability. |

The condition codes possible are

| | |
|-----|--------------------------------------------------|
| CCE | The request was granted. |
| CCG | (This condition code is not returned.) |
| CCL | The request was not granted because of an error. |

EXAMPLE:

To return information about an error reported to his program for the file identified by F7, the user can issue the following call. This returns the error-code to the word TYPE, the number of words not transferred to the word RCOUNT, and the relative number of the bad block to the double-word ERBLOCK.

FCHECK (F7,TYPE,RCOUNT,ERBLOCK)

After the intrinsic is executed, the values returned are as follows:

| Word/ Double Word | Content (Decimal Values) | Meaning |
|----------------------|----------------------------------------------------------|-----------------------------------------------------------------------------|
| <i>TYPE</i> | <i>73 (in low-order bits) 0 (in high-order bits)</i> | <i>A bounds-check violation occurred.</i> |
| <i>RCOUNT</i> | <i>53</i> | <i>Fifty-three records remain to be written, as a result of this error.</i> |
| <i>ERBLOCK</i> | <i>20</i> | <i>The 21st block was involved in the error.</i> |

DIRECTING FILE CONTROL OPERATIONS

The user can perform various control operations on a file (or the device on which it resides) by issuing the FCONTROL intrinsic call. These include: supplying a printer or terminal carriage-control directive, verifying input/output, reading the hardware status word pertaining to the device on which the file resides, setting a terminal's time-out interval, rewinding the file, writing and end-of-file indicator, and skipping forward or backward to a tape mark. The FCONTROL intrinsic applies to files on disc, tape, terminal, or line printer. (The FCONTROL intrinsic can also be used to perform various terminal functions, such as changing the terminal speed or enabling parity-checking. These applications of FCONTROL are described in Section VIII.) The FCONTROL intrinsic format is

PROCEDURE *FCONTROL (filenum,controlcode,param) ;*

VALUE *filenum,controlcode;*

INTEGER *filenum,controlcode;*

LOGICAL *param;*

OPTION EXTERNAL;

add 8-74 → 8-98

The FCONTROL call parameters are

filenum A word identifier supplying the filenumber of the file for which the control operation is performed, through SPL/3000 conventions.

controlcode An integer identifying the operation to be performed:

- 0 = *General Device Control*. The *param* parameter is transmitted to the appropriate device driver, and the value returned is transmitted to the user through the *param* parameter.
- 1 = *Line Control*. A request to send the value specified in the *param* parameter (below) to the terminal or line printer driver as a carriage-control directive. (Use line controls provided by FWRITE when directing to a disc file.)
- 2 = *Complete Input/Output*. This ensures that requested input/output has been physically completed. (Valid only for buffered files.)
- 3 = *Read Hardware Status Word*. This operation returns to the calling process the hardware status word from the device on which the file resides.
- 4 = *Set Time-Out Interval*. This code indicates that a time-out interval is to be applied to input from the terminal. If input is requested from the terminal but is not received in this interval, the FREAD request terminates prematurely with condition code CCL. The interval itself is specified (in seconds) in a word in the user's stack, indicated by *param*. If this interval is zero, any previously-established interval is cancelled, and no time-out occurs. *Controlcode 4* is ignored if the addressed file is not being read from the terminal.

- 5 = *Rewind File*. This repositions the file at its beginning, so that the next record read or written is the first record in the file. This code is valid only for files on disc and magnetic tape.
- 6 = *Write End-of-File*. This operation is used to denote the end of a file on disc or magnetic tape, and is effective only for those devices. If applied to a disc file, the operation writes a logical end-of-data indicator at the point where the file was last accessed. (The file label is also updated and written to disc.) If the file is an unlabeled magnetic tape file, a tape mark is written at the current position of the tape.
- 7 = *Space Forward to Tape Mark*. This moves the tape forward until a tape mark is encountered.
- 8 = *Space Backward to Tape Mark*. This moves the tape backward until a tape mark is encountered.
- 9 = *Rewind and Unload Tape File*. This repositions the tape file at its beginning and places the tape unit off-line.

NOTE: Control codes 0 and 3 will be rejected for spooled device-files. Control codes 5 through 9 (magnetic tape control) will be rejected for spooled :DATA tapes.

param

If *controlcode* is 1, *param* denotes a word containing the value to be transmitted to the terminal or line printer driver as a carriage-control or mode-control directive. The carriage-control directive is selected from Figure 6-3.

The mode control determines whether any carriage-control directive transmitted through the FWRITE intrinsic takes effect before printing (pre-space movement) or after printing (post-space movement). The mode control directive is selected from the octal codes %400 or %401 in Figure 6-3.

If *param* contains a mode-control directive, then a value is returned to *param* that shows the mode setting of the device as it was before the call to FCONTROL, as follows:

| Value | Meaning |
|-------|----------------|
| 0 | = Post-spacing |
| 1 | = Pre-spacing |

*at 100,101 oct 175
WITH CONTROL CODE = 1*

If *controlcode* is 4, *param* denotes a word in the user's stack that contains the time-out interval, in seconds, to be applied to input from the terminal.

If *controlcode* is 2, 5, 6, 7, 8, or 9, *param* is any variable or word identifier. This parameter is needed by FCONTROL to satisfy the internal requirements of this intrinsic. However, it serves no other purpose and is not modified by the intrinsic.

The condition codes possible are

- CCE The request was granted.
- CCG (This condition code is not returned.)
- CCL The request was denied because an error occurred.

EXAMPLES:

The user can write a tape mark on the file identified as F47 by issuing this call;

FCONTROL (F47,6,X);

He can verify that the most recently-requested FWRITE operation against this file is actually completed by entering:

FCONTROL (F47,2,X);

The programmer can transmit a mode-control code to a line-printer file (F48) by issuing the following call. (The mode-control code is found in the word PRTCN. If this code is %401, the pre-space movement option is set.) If, after the FCONTROL intrinsic is executed, a value of 1 is returned to PRTCN, this indicates that the previous mode setting was the pre-spacing option.)

FCONTROL (F48,1,PRTCN);

DECLARING ACCESS-MODE OPTIONS

The user can activate or deactivate the following access-mode options by issuing the FSETMODE call: automatic error recovery, critical output verification, and terminal control by the user. The access mode established remains in effect until another FSETMODE is issued or until the file is closed. (This intrinsic applies to files on all devices.) The format is

PROCEDURE *FSETMODE (filenum,modeflags) ;*

VALUE *filenum,modeflags;*

INTEGER *filenum;*

LOGICAL *modeflags;*

OPTION EXTERNAL;

The FSETMODE call parameters are

filenum A word identifier supplying the filenumber of the file to which this call applies, through SPL/3000 conventions.

modeflags A 16-bit value that denotes the access-mode options in effect, as detailed below. If this parameter is omitted, all bits are set to zero by default.

The bits in the *modeflags* parameter are interpreted as follows:

| Bit Nos. | Access Mode Option |
|----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| (14:1) | Critical Output Verification |
| | 1 = All output to the file is to be verified as physically complete before control returns from a write intrinsic to the user's program. For each successful logical write operation, a condition code (CCE) is returned <i>immediately</i> to the user's program. |
| | 0 = Output is not verified. |
| (13:1) | Terminal Control by the User. |
| | 1 = Inhibit normal terminal control by the system; thus, MPE/3000 will <i>not</i> issue an automatic carriage-return and line feed at the completion of each terminal input line. |
| | 0 = MPE/3000 will automatically issue the automatic carriage-return and line feed for the terminal. This parameter is ignored if the output device is not a terminal. |
| (12:1) | Tape Error Recovery |
| | 0 = Report recovered tape error with CCE. |
| | 1 = Report recovered tape error by FREAD or FWRITE with CCL and error number. |

The remaining thirteen bits (0:12) and (15:1) in this group are not used and are always set to *zero*.

The condition codes possible are

| | |
|-----|--------------------------------------------------------|
| CCE | The request was granted. |
| CCG | (This condition code is not returned.) |
| CCL | The request was not granted because an error occurred. |

EXAMPLE:

To establish an access mode where output operations are verified, terminal control is the user's responsibility, and a recovered tape error is reported with the CCE condition code, the programmer desires the following bit settings:

| | | | | | | | | | | | | | | | | |
|------------------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
| Bit No. | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| Bit Setting | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| Octal Equivalent | 0 | 0 | | 0 | | | 0 | | | 0 | | | 6 | | | |

For the file identified as IDENT, he can establish these settings by entering the octal value 0006, (preceded by the customary % required to denote an octal number) as the modeflags parameter in this FSETMODE call:

```
FSETMODE ( IDENT, %6 );
```

LOCKING AND UNLOCKING FILES

Sometimes the user may want to manage the sharing of a file between two or more processes so that no two of these processes can access the file simultaneously. He can do this for files on any devices through the dynamic locking and unlocking of files with the FLOCK and FUNLOCK intrinsic calls. Because such locking and unlocking entails the use of resource identification numbers (RIN's), however, discussion of the FLOCK and FUNLOCK calls is deferred until the discussion of RIN's in Section IX.

RENAMING A FILE

The user can rename (change the actual designator of) an open disc file, with the FRENAME intrinsic call. This effectively changes the actual designator (including lockword) of the file. This file must be either:

1. A new file, or
2. An existing file, opened for fully-exclusive access, to which the user has write-access (specified by the security provisions of the file).

The format of FRENAME is

PROCEDURE

FRENAME (filenum, newfilereference) ;

VALUE filenum;

INTEGER filenum;

BYTE ARRAY newfilereference;

OPTION EXTERNAL;

The parameters are

| | |
|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>filenum</i> | A word identifier supplying the filenumber of the file to be renamed, through SPL/3000 conventions. |
| <i>newfilereference</i> | A byte-array containing an ASCII string representing the new name for the file. This parameter may not specify the name of an account other than that to which the file is currently assigned. Additionally, the caller must have <i>save</i> access to the group specified or implied by the string. The string must begin with a letter, contain alphanumeric characters, and terminate with a non-alphanumeric character other than a slash or period. |

The condition codes possible are

| | |
|-----|--------------------------------------------------------|
| CCE | The request was granted. |
| CCG | (This condition code is not returned.) |
| CCL | The request was not granted because an error occurred. |

EXAMPLE:

To rename a file identified as *FNR*, with the new designator contained in the byte-array *NEWFL*, the user can enter:

FRENAME (FNR,NEWFL);

DETERMINING INTERACTIVE AND DUPLICATIVE FILE PAIRS

An input file and a list file are said to be *interactive* if a real-time dialogue can be established between a program and a person using the list file as a channel for programmatic requests, with appropriate responses from a person using the input file. For example, an input file and a list file opened to the same teletype terminal (for a session) would constitute an interactive pair. An input file and a list file are said to be *duplicative* when input from the former is automatically duplicated on the latter. The user can determine whether a pair of files is interactive or duplicative through the *FRELATE* intrinsic call. (The interactive/duplicative attributes of a file pair do not change between the time it is opened and the time it is closed.) (This intrinsic applies to files on all devices.)

The format of *FRELATE* is

LOGICAL PROCEDURE

FRELATE (infilenum,listfilenum) ;

VALUE *infilenum,listfilenum;*

INTEGER *infilenum,listfilenum;*

OPTION EXTERNAL;

This intrinsic returns to the calling process a word showing whether the files are interactive and/or duplicative.

The parameters are

| | |
|--------------------|---------------------------------------------------------------------------------------------|
| <i>infilenum</i> | A word identifier supplying the filenumber of the input file, through SPL/3000 conventions. |
| <i>listfilenum</i> | A word identifier supplying the filenumber of the list file, through SPL/3000 conventions. |

The word returned to the user's process contains two significant bits:

| | | |
|-------------------|---|-------------------------------------------------------------------------------|
| <i>Bit (15:1)</i> | = | 1, if <i>infilenum</i> and <i>listfilenum</i> form an <i>interactive</i> pair |
| | = | 0, if they do not. |
| <i>Bit (0:1)</i> | = | 1, if <i>infilenum</i> and <i>listfilenum</i> form a <i>duplicative</i> pair. |
| | = | 0, if they do not. |

Bits 1-14 are not used, and are always set to zero.

The condition codes possible are

| | |
|-----|--------------------------------------------------------|
| CCE | The request was granted. |
| CCG | (This condition code is not returned.) |
| CCL | The request was not granted because an error occurred. |

EXAMPLE:

To determine whether the files identified as FABLE and FBAKER are interactive or duplicative, the user can issue this call:

INDUP := FRELATE (FABLE,FBAKER);

After the intrinsic is executed, if Bit 15 of the word INDUP is set to 0 but Bit 0 of that word is set to 1, the user knows that FABLE and FBAKER are not interactive but are duplicative.

FILE INTRINSIC FILE-TYPE SUMMARY

The file intrinsics described in this section and the types of files/devices to which they apply are summarized below:

| All Devices | Disc | Disc and Tape | Printer, Terminal, Tape, and Disc |
|-------------|-------------|---------------|--------------------------------------|
| FOPEN | FUPDATE | FSPACE | FCONTROL |
| FCLOSE | FPOINT | | |
| FREAD | FREADDIR | | |
| FWRITE | FWRITEDIR | | |
| FCHECK | FREADSEEK | | |
| FGETINFO | FRENAME | | |
| FSETMODE | FREADLABEL | | |
| FLOCK | FWRITELABEL | | |
| FUNLOCK | | | |
| FRELATE | | | |



SECTION VII

Managing Program Libraries

The user can perform various management functions relating to program libraries. In general, he can

- Modify user subprogram libraries (USL's) by adding, deleting, activating, and de-activating relocatable binary modules (RBM's) stored in them
- Manage libraries used to resolve program references to external procedures
- Dynamically attach and detach a library procedure to a running program

The first two of these functions are performed through the MPE/3000 Segmenter.

MANAGING USER SUBPROGRAM LIBRARIES (USL'S)

As noted previously, compiled user programs are stored in files called USL's that reside on disc. In any particular USL, each compiled program unit exists as a Relocatable Binary Module (RBM). (In FORTRAN/3000, a program unit is a main program, subroutine, function, or block data unit; in SPL/3000, a program unit is an outer block or procedure. In COBOL/3000, a program unit is a main program or section.)

To prepare a program (and the program units it references) for execution, the MPE/3000 Segmenter selects the appropriate RBM's from the USL and binds them into linked segments written on a program file.

Each RBM is identified by the symbolic name (label) of the primary entry point of the program unit stored in the RBM; the RBM may be associated with a segment name, determined during compilation, that identifies the segment to which it will belong. An activity bit is associated with each entry point (the primary entry point and any secondary entry points). This activity bit determines whether the program unit currently can be entered at the corresponding entry point. When a compiler writes an RBM on a USL, all entry points in the RBM are set to the *active* (entry allowed) state. Through the MPE/3000 Segmenter, they can then be switched from that state to the *inactive* (entry disallowed) state, and back again, by the user.

When a program file is prepared from the USL, all RBM's having at least one active entry point are extracted from the USL for segmentation on the program file. Those having identical segment names are placed in the same segment. To permit the creation of a program file that can be executed properly, only one outer-block or main-program RBM can have active entry points, along with the RBM's for the subprograms or procedures on the USL that it references.

Entry points having the same name are permitted in a USL. Each of these is uniquely identified by its name used in conjunction with an index integer; this integer denotes the particular definition of the entry point in chronological terms, beginning with one to indicate the most recent definition. Since the name of an RBM is equivalent to that of its primary entry-point, the name/index identification method allows RBM's containing different versions of the same program unit to be referenced by the same name used with different index integers. Thus, to identify the most recent version of a program unit in an RBM named START, the user would specify START (1); to indicate the next most recent version, he would specify START (2). A special convention (index = 0) is used to indicate the most recent *active* definition; for example, START (0).

Sometimes, the user may want to alter the structure of a program to incorporate new features or modify existing ones. For example, the user may want to include a new version of a main program or subroutine in place of a previous version. He can do this by adding and deleting RBM's in the pertinent USL, or by activating or deactivating appropriate RBM entry points.

Additionally, the user may want to change the segmentation of a program (and its associated subprograms) to improve its efficiency. In some operating systems, such a modification would require recompilation. Under MPE/3000, however, the user can simply re-arrange the RBM's and then prepare the USL into a new program file.

Before modifying a USL, the user first accesses the MPE/3000 Segmenter by entering:

`_SEGMENTER [listfile]`

listfile AN ASCII file (formal designator SEGLIST) from the output set, to which is written any listable output generated by Segmenter subsystem commands. (The designator SEGLIST should *not* be used as the *actual* file designator.) If *listfile* is omitted, the standard job/session list device (\$STDLIST) is assigned. (Optional parameter.)

He then enters any of the Segmenter commands defined below to modify the USL. In a session, a dash is issued as the prompt character for each Segmenter command. No prompt character, however, is allowed in a batch job. (The rules covering delimiters, blanks, positional parameters, and continuation characters in Segmenter commands are the same as those rules for all other MPE/3000 commands.) When the user has completed USL modification, he enters the following command to return control from the Segmenter back to the MPE/3000 command interpreter:

`_EXIT`

Segmenter commands, rather than the conventional MPE/3000 commands, are required to create and change USL's because these files have a special format. However, once a USL has been created, it can be managed by other MPE/3000 commands and intrinsics — for example, it can be purged (:PURGE command), renamed (:RENAME command), or read (FREAD intrinsic).

Error messages that may occur while using the Segmenter are listed in Figure 10-7 in Section X.

Creating New USL's

To create a new (job temporary) USL onto which RBM's containing user program units can be compiled or copied, the programmer enters a command of this format. The USL then exists as a job temporary file.

-BUILDUSL filereference,records,extents

filereference The name (file designator) assigned to the new USL file. (This is a fully-qualified file reference that may include file name, group name, and account-name, plus a lockword.) (Required parameter.)

records The length of this file in terms of 128-word logical records. (Required parameter.) NOTE: All logical records in files referenced by the Segmenter command parameters described in this section of the manual are 128-words long.

extents The number of disc extents to be allocated to the file. (Required parameter.)

EXAMPLE:

To create a new USL named NEWUSL, the user enters the following command. The USL will contain 300 logical records and will be allocated one extent.

```
-BUILDUSL NEWUSL,300,1
```

Designating USL's For Management By The User

To enter commands that modify an existing USL in any way, the user must first identify this USL by the -USL command:

-USL filereference

filereference The name (and optional group and account names) of the USL file to be manipulated. (Required parameter.)

Then, all subsequent USL-modification commands will refer implicitly to this USL until a new -USL command is issued.

If, however, the user has entered a -BUILDUSL command, this acts as an implicit -USL command for the file referenced. That is, all USL-modification commands between a -BUILDUSL command and the next -USL command or -BUILDUSL command refer to the file specified in the first -BUILDUSL command.

EXAMPLE:

The following sample terminal listing shows how different USL files are designated for management by Segmenter commands:

| Command | Operation |
|----------------------------------------------|--------------------------------------------------------------------------------------------------------------|
| · · · :SEGMENTER LSTFL | Accesses the Segmenter. |
| -USL FILE1 | Designates the USL file FILE1 (in the user's log-on group and account) for management. |
| -USE UNIT, PROG(2) -CEASE UNIT, PROG(1) } | These commands modify FILE1. |
| -BUILDUSL FILE2, 200, 1 | Creates a new file, FILE2 (in the user's log-on group and account), and designates this file for management. |
| -AUXUSL FILE1 -COPY UNIT, ORBM } | Copies an RBM from FILE1 to FILE2. |
| -USL FILE3 | Designates FILE3 (in the user's log-on group and account) for management. |
| -LSTUSL | Lists RBM's residing in FILE3. |
| · · · -EXIT | Exits from Segmenter. |

Activating Entry Points

To activate one or more program unit (RBM) entry points in the USL currently designated for management, the user enters the USE command:

```
_USE [pointspec,] name [(index)]
```

pointspec Is ENTRY, to activate the entry point indicated by *name (index)*; or
 UNIT, to activate all entry points in the RBM indicated by (the primary entry point) *name (index)*; or
 SEGMENT, to activate all entry points in all RBM's associated with the segment *name*.

 The default value is ENTRY. (Optional parameter.)

name The name of the entry point, RBM (referred to by its primary entry-point name), or segment referenced. (Required parameter.)

index The index integer further specifying the entry point *name*. The index may be used when the USL contains more than one entry point of this name. If *index* is omitted, a default value of 0 is assigned. If *pointspec* is SEGMENT, the *index* parameter does not apply. (Optional parameter.)

EXAMPLE:

Suppose the user is managing a USL file named FILE2. On this file, he wants to activate all entry points in the most recent active version of the program unit whose primary entry point (RBM name) is RB20.

```
_USL FILE2  
    .  
    .  
    .  
▶ _USE UNIT, RB20
```


Deactivating Entry Points

To deactivate one or more entry points in the USL currently designated for management, the user enters:

-CEASE [pointspec,] name [(index)]

pointspec Is ENTRY, to deactivate the entry point indicated by *name (index)*; or

UNIT, to deactivate all entry points in the RBM indicated by *name (index)*; or

SEGMENT, to deactivate all entry points in all RBM's associated with segment *name*.

The default parameter is ENTRY. (Optional parameter.)

name The name of the entry point, RBM, or segment referenced. (Required parameter.)

index The index integer further specifying the entry point *name*. The index may be used when the USL contains more than one entry point or RBM of this name. If *index* is omitted, a default value of 0 is assigned. If *pointspec* is SEGMENT, the *index* parameter does not apply. (Optional parameter.)

EXAMPLE:

To deactivate the endpoint *BEGIN2* in the USL named *FILE3*, the user enters:

```
➡ -USL FILE3  
  .  
  .  
  .  
  .  
  .  
  .  
  .  
  .  
  .  
  .  
  .
```

Deleting RBM's

To delete one or more RBM's from a USL, the user can enter the **-PURGERBM** command:

```
-PURGERBM [rbmspec,] name [(index)]
```

rbmspec Is **UNIT**, to delete the RBM identified by *name* (*index*); or
SEGMENT, to delete all RBM's associated with the segment *name*.

The default parameter is **UNIT**. (Optional parameter.)

name If *rbmspec* is **UNIT**, this is the name of the RBM to be deleted. If
rbmspec is **SEGMENT**, this is the name of the segment in which all
RBM's are to be deleted. (Required parameter.)

(*index*) If *rbmspec* is **UNIT**, this is the index integer further specifying the
RBM name; the index may be used when more than one RBM of this
name exists in the USL. If *index* is omitted, a default value of 0 is
assigned. If *rbmspec* is **SEGMENT**, this parameter does not apply.
(Optional parameter.)

EXAMPLE:

To delete the RBM named *XPOINT* (on the USL named *XFILE*), the user enters:

```
-USL XFILE  
.  
.  
.  
▶ -PURGERBM UNIT, XPOINT
```

Assigning New Segment Names to RBM's

The user can change the segment name associated with an RBM, thus assigning the RBM to a different code segment the next time it is prepared onto a program file. The segment name is changed by entering:

```
_NEWSEG newsegname ,rbmname[(index)]
```

newsegname The new segment name to be associated with the RBM. (Required parameter.)

rbmname The name of the RBM whose segment name is to be changed. (Required parameter.)

(index) The index integer further specifying the RBM name. It may be used when more than one RBM of the same name exists in the USL. If *index* is omitted, a default value of 0 is assigned. (Optional parameter.)

EXAMPLE:

To associate the new segment name *NEWNAME* with an RBM named *RB3* in the USL *FILE7*, the user enters:

```
_USL FILE7  
.  
.  
.  
▶ _NEWSEG NEWNAME, RB3  
.  
.  
.
```

Transferring RBM's

The user can transfer one or more RBM's from another USL to the currently-managed USL. First, he must designate the USL source (from which the RBM's are transferred), by entering the `-AUXUSL` command:

`-AUXUSL filereference`

filereference The name (and optional group and account names) of the USL file from which the RBM's are to be transferred. (Required parameter.)

Next, the user transfers (copies) the RBM's to the destination USL by entering the `-COPY` command. (All subsequent `-COPY` commands will refer implicitly to the source USL file designated by the current `-AUXUSL` command, until a new `-AUXUSL` command is encountered.)

`-COPY [rbmspec,] name [(index)]`

rbmspec Is `UNIT`, to transfer the RBM identified by *name (index)* from the source USL; or

 `SEGMENT` to transfer *all* RBM's associated with the segment *name* from the source USL.

The default parameter is `UNIT`. (Optional parameter.)

name If *rbmspec* is `UNIT`, *name (index)* identifies the RBM to be transferred. If *rbmspec* is `SEGMENT`, *name* identifies the segment from which all RBM's are transferred. (Required parameter.)

index The index integer further specifying the RBM name; the index may be used when more than one RBM of this name exists in the USL. If *index* is omitted, a value of 0 is assigned by default. This parameter is ignored if *rbmspec* is `SEGMENT`. (Optional parameter.)

When an RBM is transferred, the segment name associated with it (if any) remains the same.

Once the operation is finished, the RBM's in the new segment are found in the reverse order from that in which they were found in the original segment.

EXAMPLE:

To move all RBM's associated with the segment name *SEGNAME* from the USL named *BASEFL* to the USL named *DESTFL*, the user enters:

```
  _USL DESTFL  
  .  
  .  
  .  
  _AUXUSL BASEFL  
  .  
  .  
  .  
▶ _COPY SEGMENT, SEGNAME
```

Listing RBM's

The user can list the names of the RBM's in the currently-managed USL, their segment names, entry points, and other related information, by entering:

```
  _LISTUSL
```

The output is written on the file designated in the *listfile* parameter of the *:SEGMENTER* command. It shows all program unit entry points. Those entry points related to procedure or block data program units are listed under the segment in which they are contained. Interrupt procedures and block data program units appear in separate lists.

An example of the output produced by the -LISTUSL command is shown in Figure 7-1. On this listing, significant entries are indicated with arrows, keyed to the discussion below. All numbers appearing on this listing are octal values.

| Item No. | Meaning |
|----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1 | The name of the USL file (filename.groupname.accountname). |
| 2 | The segment name. |
| 3 | The RBM (entry-point or program unit) name. |
| 4 | The number of words (octal) in the RBM code module. |
| 5 | The type of entry-point named, where: P = Procedure primary entry point. SP = Secondary entry point for procedure. OB = Outer block primary entry point. SO = Secondary entry point for outer block. BD = Block data entry point. In = Interrupt procedure entry point (n is the interrupt procedure type number, ranging from 0 through 2). |
| 6 | The active/inactive indicator, where: A = Active I = Inactive |
| 7 | The callability indicator, where: C = The entry point is directly callable by the user. U = The entry point is not directly callable by the user. |
| 8 | The privileged/not-privileged indicator: P = Privileged. N = Not privileged. |
| 9 | The internal flag status: H = Internal flag <i>on</i> R = Internal flag <i>off</i> . (The internal flag and its purpose are explained later in this section.) |
| 10 | The total size of the USL file, in words (octal). |

| Item No. | Meaning |
|----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 11 | The number of words (in octal) used in the directory portion of the USL file. |
| 12 | The number of words (in octal) in the directory part of the USL file, filled with meaningless information, as a result of various RBM manipulations. |
| 13 | The number of unused words (in octal) in the directory portion of the USL file. |
| 14 | The number of words (in octal) used in the RBM-information portion of the USL file. |
| 15 | The number of words (in octal) in the RBM-information portion of the USL file, filled with meaningless information, as a result of various RBM manipulations. |
| 16 | The number of unused words (in octal) in the RBM-information portion of the USL file |

Preparing Program Files

Once the user has modified a USL, he can prepare the active RBM's residing on it into a program file. He can do this by entering the `:PREP` command, described in Section IV. MPE/3000 also permits him to enter a similar command (`-PREPARE`) through the Segmenter, in which case the *dash* (rather than the colon) is used as the prompt character (in sessions) and the parameters defined below apply. The currently-managed USL is used as the RBM source.

`-PREPARE progfile`

```

[;ZERODB]
[;PMAP]
[;MAXDATA=segsz]
[;STACK=stacksz]
[;DL=dlsz]
[;CAP=caplist]
[;RL=filename]

```

progfile The name of the program file onto which the prepared program segments are to be written. This can be any binary file from the output set. (Required parameter.)

NOTE: Code segments in a program file cannot lie across disc extent boundaries. Thus, all segments in such files must be constructed within one extent.

USL FILE SCR4.PUB.SYS

| | | | | | | | | |
|-----------------|--------|------|-------------|--------|------|-----|-----|-----|
| SEG40 | ← 2 | ← 1 | ← 4 | ← 5 | ← 6 | ← 7 | ← 8 | ← 9 |
| LISTRL | 351 | P | A | C | N | R | | |
| CLEANUPRLEXTNBU | 13 | F | A | C | N | R | | |
| GETNEXTREXTN | 34 | F | A | C | N | R | | |
| SETUPRLEXTNBUF | 36 | F | A | C | N | R | | |
| ADDEDPROC | 27 | F | A | C | N | R | | |
| DELETEDPROC | | SF | A | C | | R | | |
| FIXUPRL | 502 | P | A | C | N | R | | |
| REMOVERL | 72 | P | A | C | N | R | | |
| INSERTRL | 434 | F | A | C | N | R | | |
| CLEANUPRLBUF | 11 | F | A | C | N | R | | |
| DELETERLENTY | 46 | F | A | C | N | R | | |
| FINDRLDIRSPACE | 72 | F | A | C | N | R | | |
| GETNEXTRENTY | 44 | F | A | C | N | R | | |
| SETUPRLBUF | 20 | F | A | C | N | R | | |
| RENTYPARMS | 23 | F | A | C | N | R | | |
| GETNEXTRECD | 20 | F | A | C | N | R | | |
| SEARCHRL | 76 | F | A | C | N | R | | |
| SAVERLMAP | 11 | F | A | C | N | R | | |
| GETRLMAP | 14 | F | A | C | N | R | | |
| RETURNRLSPACE | 27 | F | A | C | N | R | | |
| FINDRLSPACE | 126 | F | A | C | N | R | | |
| SEG33 | | | | | | | | |
| REMOVESL | 106 | F | A | C | N | R | | |
| BINDSEGS | 415 | F | A | C | N | R | | |
| FIXUPSL | 125 | F | A | C | N | R | | |
| RETURNSSLSPACE | 14 | F | A | C | N | R | | |
| SEG32 | | | | | | | | |
| INSERTNEWSEG | 202 | F | A | C | N | R | | |
| MAKEENTRIES | 163 | F | A | C | N | R | | |
| INSERTSEG | 203 | F | A | C | N | R | | |
| FINDSLSPACE | 220 | F | A | C | N | R | | |
| FINDSLRECORD | 35 | F | A | C | N | R | | |
| SEG31 | | | | | | | | |
| CLEANUPRTBUF | 11 | F | A | C | N | R | | |
| GETREFTABENTY | 33 | F | A | C | N | R | | |
| ----- | | | | | | | | |
| SEG1 | | | | | | | | |
| CB* | 606 | OB | A | C | N | | | |
| CORRECTCLASS | 32 | F | A | C | N | R | | |
| SETACTIVITY | 43 | P | A | C | N | R | | |
| FILE SIZE | 400000 | ← 10 | | | | | | |
| DIR. USED | 10001 | ← 11 | INFO USED | 40737 | ← 14 | | | |
| DIR. GARB. | 0 | ← 12 | INFO GARB. | 0 | ← 15 | | | |
| DIR. AVAIL. | 67577 | ← 13 | INFO AVAIL. | 237041 | ← 16 | | | |

Figure 7-1. -LISTUSL Command Output

| | |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>ZERODB</i> | An indication that the initially-defined DL-DB area, and uninitialized portions of the DB-Q (initial) area will be initialized to zero. If this parameter is omitted, these areas are not affected. (Optional parameter.) |
| <i>PMAP</i> | An indication that a listing describing the prepared program will be produced on the device specified by the :SEGMENTER command parameter <i>listfile</i> . (An example of this listing appears under the discussion of the :PREP command in Section IV.) If this parameter is omitted, no listing is produced. (Optional parameter.) |
| <i>segsiz</i> | Maximum stack area (Z-DL) size permitted, in <i>words</i> . This parameter is included if the user expects to change the size of the DL-DB or Z-DB areas during process execution. If omitted, MPE/3000 assumes that he will not change these areas. (Optional parameter.) |
| <i>stacksiz</i> | The size of the user's initial local data area (Z-Q (initial)) in the stack, in <i>words</i> . This overrides the <i>stacksiz</i> estimated by the Segmenter, which applies if the <i>stacksiz</i> parameter is omitted. (The default is a function of estimated stack requirements for each program unit in the program. Since it is difficult for the system to predict the behavior of the stack at run time, the user may want to override the default by supplying his own estimate with <i>stacksiz</i> .) (Optional parameter.) |
| <i>dlsiz</i> | The DL-DB area to be initially assigned to the stack. This area is mainly of interest only in programmatic applications, and is discussed in detail in Section II. If the <i>dlsiz</i> parameter is omitted, a value of zero is used. (Optional parameter.) |
| <i>caplist</i> | The <i>capability-class attributes</i> associated with the user's 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
- BA = Local batch access
- PH = Process handling
- DS = Data segment management
- MR = Multiple resource management
- PM = Privileged-mode operation

The user who issues the -PREPARE command can only specify capabilities that he himself possesses (through assignment by the Account Manager). If the user does not specify any capabilities, only IA and BA (if the user possesses them) will be assigned to this program. (Optional parameter.)

filename The name (and optional account and group names) of a relocatable procedure library (RL) to be searched to satisfy external references during preparation. This can be any permanent file of type RL. It need not belong to the 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 prepared. If *filename* is omitted, no library is searched. (Optional parameter.)

For the *stacksize*, *dsize*, and *maxdata* parameters, a value of -1 denotes the default (equivalent to omitting the parameter).

USING EXTERNAL PROCEDURE LIBRARIES

As part of its file domain, each group within an account may optionally include files comprising procedure *libraries*. These libraries contain external procedures frequently called by programmers accessing this group. They permit several users to concurrently access common routines with efficiency.

Under MPE/3000, there are two types of libraries: Relocatable Libraries and Segmented Libraries. Each group optionally may have one or more Relocatable Libraries and one Segmented Library.

Relocatable Libraries

Relocatable Libraries (RL's), are specially-formatted files that are searched at *program preparation* time to satisfy references to external procedures required by the user's program. Within such libraries, these procedures exist in RBM form (as they would on a USL). When a program is prepared, these procedures are placed in a single segment and linked to the user's program in the resulting program file. Since a segment containing procedures from an RL is specifically constructed for a given program, any such segment cannot be shared concurrently by different programs. Instead, individual copies of the required procedures must be made and segmented for each program requesting them.

An RL is always a permanent file, but it need not belong to the user's log-on group nor have a reserved local file name.

The RL to be searched during preparation of the user's program is specified by the *filename* parameter of the :PREP, :PREPRUN, or -PREPARE command. For this parameter, the *filereference* format introduced in Section V is used.

Procedures in an RL may not contain TRACE variables.

EXAMPLE:

To specify that an RL named RL in the group GROUP2 of the user's log-on account be searched during preparation of a program from the USL ALPHA1 to the program file ALPHA2, the user enters the following command:

:PREP ALPHA1, ALPHA2; RL=RL.GROUP2

Segmented Libraries

Segmented Libraries (SL's) are specially-formatted files that are searched at program-allocation time to satisfy references to external procedures. These libraries, like program files, contain procedures in segmented (prepared) form. An individual procedure may be the only procedure in its segment, or may exist in a segment containing many other procedures. When a procedure is referenced, the segment containing it is loaded along with other segments referenced by that procedure. Since the segmentation is not altered when different programs reference procedures in an SL, these procedures may be shared concurrently by many programs.

From one to three SL's may be searched during allocation of the user's program. These are

1. The Group Library (GSL), which is actually the library of the group under which the program file is stored. This library is readable by any user who can access this group.
2. The account's Public Library (PSL), which is the library of the public group of the account under which the program file is stored. This library is readable by any user who can access this account.
3. The System Library (SSL), which is the library of the public group of the system account. This library can be accessed by all users of the system.

The order in which the libraries are searched is specified by the LIB parameter in the :RUN command that requests program allocation/execution, as follows:

| LIB Parameter | Specification |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| LIB = G | The libraries are searched in this order: <ul style="list-style-type: none">● Group Library● Public Library● System Library |
| LIB = P | The libraries are searched in this order: <ul style="list-style-type: none">● Public Library● System Library |
| LIB = S | Only the System Library is searched. |

When no LIB parameter is specified, the default is LIB = S.

EXAMPLE:

To specify that the public library and then the system library be searched during allocation, the user enters the following command when executing the program ALPHA2.

`:RUN ALPHA2; LIB=P`

The name of the System Account is always SYS. Within an account, the name of any public group is always PUB; the name of the segmented library is always SL. Thus, when referencing a library file in an MPE/3000 command or intrinsic, the following rules apply.

1. *If the user's program file is a permanent file, with the name X.Y.Z (in the filereference format), then:*

GSL is referenced as SL.Y.Z

PSL is referenced as SL.PUB.Z

SSL is referenced as SL.PUB.SYS

If Y.Z = PUB.SYS, then only LIB = S is a legal entry in the :PREPRUN or :RUN command.

If Y.Z = PUB.Z, then LIB = P or LIB = S are legal entries in the :PREPRUN or :RUN command.

During allocation, the proper libraries are determined by the position of the program file in the file system directory.

2. *If the user's program file is not a permanent file, but is a job/session temporary or passed file with the name X.Y.Z, where Y is the log-on group and Z is the log-on account, or is nameless (such as the file \$OLDPASS) then:*

GSL = SL.Y.Z

PSL = SL.PUB.Z

SSL = SL.PUB.SYS

In the above cases, LIB = G, LIB = P, or LIB = S are all legal entries in the :PREPRUN or :RUN command.

During allocation, the proper libraries are determined by the log-on group and account.

As an example of how SL's are used, procedures called frequently by many users throughout the system are stored in the SSL. Those used frequently throughout an account can be stored in an PSL. Those used by only a few users in a single group, less frequently, might be stored in a GSL.

Because the global area of the user's stack is already formed at the time a user's program references an external procedure in an SL, the library procedures cannot contain global variables. From the user's standpoint, this means that SPL/3000 procedures in a library cannot contain TRACE, EXTERNAL or OWN variables. Also, FORTRAN procedures in a library cannot contain: DATA, COMMON, LABELED COMMON, or TRACE variables; FORMAT statements; or references to LOGICAL UNITS.

CREATING AND MAINTAINING RELOCATABLE LIBRARIES (RL'S)

To enter commands to create and maintain relocatable libraries, the user first accesses the Segmenter by entering the :SEGMENTER command. He may then enter any of the following commands, preceded (in sessions) by a dash as the prompt character.

As with USL's, Segmenter commands are required to create and change RL's because these files are written in a special format. But once an RL is created, it can be referenced by other MPE/3000 commands and intrinsics; for example, it is purged with the :PURGE command, renamed with the :RENAME command, or read with the FREAD intrinsic.

Creating a Relocatable Procedure Library (RL) File

To create a permanent, formatted RL file, the user enters the following command.

_BUILDRL filereference, records, extents

| | |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>filereference</i> | The filename of the new RL, in the <i>filereference</i> format, (optionally including group and account identifiers). (Required Parameter.) |
| <i>records</i> | The total maximum file capacity, specified in terms of 128-word, binary logical records. (Required parameter.) |
| <i>extents</i> | The total number of disc extents that can be dynamically-allocated to the file as logical records are written to it. The size of each extent is determined by the <i>records</i> parameter value divided by the <i>extents</i> parameter value. The <i>extents</i> value must be an entry from 1 to 16. (Required parameter.) |

Designating RL's for Management by the User

To enter commands that modify an existing RL in any way, the user must first identify the RL by the `-RL` command:

`-RL filereference`

filereference The name (and optional group and account names) of the RL to be modified, in the *filereference* format. (Required parameter.)

Then, all subsequent RL-modification commands will refer implicitly to this RL until a new `-RL` command is issued. If, however, the user has entered a `-BUILDRL` command, this acts as an implicit `-RL` command for the file referenced. That is, all RL-modification commands between a `-BUILDRL` command and the next `-RL` command or `-BUILDRL` command refer to the file specified in the first `-BUILDRL` command.



Adding a Procedure to an RL

To add a procedure to the currently-managed RL, the user copies the RBM containing that procedure from the currently-managed USL to the RL. The user must have write-access to the designated RL file. The procedure is added with the following command:

`-ADDRL name[(index)]`

name The name (primary entry point) of the RBM containing the procedure to be added to the RL. (Required parameter.)

index The index integer further identifying the RBM. The index may be used when the currently-managed USL contains more than one active RBM of this name. If *index* is omitted, a value of 0 is assigned by default. (Optional parameter.)

Deleting an Entry-Point or Procedure from an RL

To delete an entry point of a procedure, or an entire procedure, from an RL, the user enters:

`-PURGERL [rlspec,] name`

rlspec Is `UNIT`, to delete the procedure identified by *name*; or

`ENTRY`, to delete the entry point identified by *name*.

The default parameter is `ENTRY`. (Optional parameter.)

name If *rlspec* is `UNIT`, this is the name of the procedure to be deleted. If *rlspec* is `ENTRY`, this is the name of the entry point to be deleted. (Required parameter.)

When the last entry point in a procedure is deleted, the entire body of code representing that procedure is deleted.

Additionally, the entire RL can be deleted by the MPE/3000 command, :PURGE.

Listing Procedures in an RL

To list all procedure entry-points and external references in the currently-managed RL, the following command is used:

`-LISTRL`

An example of the output produced by the `-LISTRL` command is shown in Figure 7-2. On this listing, significant entries are indicated with arrows, keyed to the discussion below. (All numbers appearing on this listing are octal values.)

| Item No. | Meaning |
|----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1 | The name of the RL file (filename.groupname.accountname). |
| 2 | The procedure entry-point name. |
| 3 | The parameter checking level for the entry point. |
| 4 | The entry-point address (relative displacement within the code module). |
| 5 | The RL file address of the procedure information block. |
| 6 | The number of entry points in the procedure. (If this field is blank, the entry-point name field (Item 2) names a secondary entry point; if this field contains a number, the entry-point name field contains a primary entry-point name.) |
| 7 | The length of the code module, in words. (If this field is blank, the entry-point name field names a secondary entry-point; if this field contains a number, the entry-point name field contains a primary entry-point name.) |
| 8 | The length of the procedure information block, in words. (If this field is blank, the entry-point name field names a secondary entry point; if this field contains a number, the entry-point name field contains a primary entry-point name.) |
| 9 | The name of the external reference. |
| 10 | The parameter checking level for the external reference. |
| 11 | The entry-point address of the external reference. |
| 12 | The RL file address of the external procedure information block. (If a number appears in this field <i>and</i> in the entry-point address field (Item 11), the external reference is satisfied within the RL. However, if these fields are blank, the external reference is <i>not</i> satisfied within the RL.) |
| 13 | The number of words presently used in the RL file. |
| 14 | The number of words presently available in the RL file. |

RL FILE SCR1.MPE.SYS

* ENTRY POINTS *

| | | | | | | |
|--------------|---|------|------|---|------|------|
| FERROR | 0 | 0 | 3600 | 1 | 105 | 174 |
| FREADMR | 0 | 20 | 4600 | | | |
| FEOF | 0 | 0 | 5400 | 1 | 13 | 41 |
| PRINTWARNING | 0 | 25 | 400 | | | |
| CLEARLINE | 0 | 0 | 2700 | 1 | 10 | 22 |
| FREADDIR | 0 | 0 | 700 | 1 | 12 | 53 |
| MESSAGE | 0 | 1104 | 1400 | 1 | 1210 | 1222 |
| FREADMR | 0 | 0 | 4600 | 2 | 32 | 105 |
| PRINTERROR | 0 | 0 | 400 | 2 | 122 | 235 |
| DNTOA | 0 | 0 | 2640 | 1 | 24 | 36 |
| ERROR | 0 | 0 | 640 | 1 | 5 | 34 |

* EXTERNALS *

| | | | |
|------------|---|------|------|
| GETJCW | 0 | | |
| SETJCW | 0 | | |
| DNTOA | 0 | 0 | 2640 |
| MESSAGE | 0 | 1104 | 1400 |
| PRINT | 0 | | |
| CLEARLINE | 0 | 0 | 2700 |
| PRINTERROR | 0 | 0 | 400 |
| FREADDIR | 0 | | |
| FERROR | 0 | 0 | 3600 |
| FCHECK | 0 | | |
| MESSAGE | 0 | 1104 | 1400 |
| PRINTERROR | 0 | 0 | 400 |
| QUIT | 0 | | |
| FEOF | 0 | 0 | 5400 |
| FREADDIR | 0 | | |
| FERROR | 0 | 0 | 3600 |
| FGETINFO | 0 | | |

USED

5600

AVAILABLE

23200

Figure 7-2. -LISTRL Command Output

Examples of Relocatable Library Management Commands

In the following listing, several commands that manage relocatable libraries are illustrated:

EXAMPLE:

| Command | Operation |
|-----------------------------------------|--------------------------------------------------------------------------------------------------------------|
| <code>:_SEGMENTER</code> | <i>Accesses the Segmenter</i> |
| <code>:_BUILDRL R1.G1.A1, 500, 1</code> | <i>Creates an RL file, named R1.G1.A1. The RL can contain 500 records, and is allocated one disc extent.</i> |
| <code>:_USL XFILE</code> | <i>Designates the USL file XFILE for management.</i> |
| <code>:_ADDRL XBM</code> | <i>Adds an RBM (containing one procedure) named XBM from XFILE to the RL file R1.G1.A1.</i> |
| <code>:_RL R2.G1.A1</code> | <i>Designates the RL file R2.G1.A1 for management.</i> |
| <code>:_PURGERL UNIT, PROCA</code> | <i>Deletes a procedure named PROCA from the RL file R2.G1.A1.</i> |
| <code>:_EXIT</code> . . . | <i>Exits from Segmenter, returning control to the MPE/3000 Command Interpreter.</i> |

CREATING AND MAINTAINING SEGMENTED LIBRARIES (SL'S)

To enter commands to create and maintain segmented libraries, the user first accesses the Segmenter by entering the `:SEGMENTER` command. He may then enter any of the following commands, preceded by a dash (in sessions) as the prompt character. As with USL's and RL's, Segmenter commands are required to create and change SL's because these files are written in a special format. But once an SL is created, it can be referenced by other MPE/3000 commands and intrinsics.

Creating a Segmented Procedure Library (SL) File

To create a permanent, formatted SL file, the user enters the following command. The user must have *save-access* to the group to which he assigns an SL. The SL will be created with default file-access security.

`_BUILDSL filereference, records, extents`

| | |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>filereference</i> | Any file whose local name is SL, in the <i>filereference</i> format. (The user can create an SL file with a local name other than SL, but such a file cannot be searched by the <code>:RUN</code> command or the <code>CREATE</code> or <code>LOADPROC</code> intrinsics.) (Required parameter.) |
| <i>records</i> | The total maximum file capacity, specified in terms of 128-word binary logical records. (Required parameter.) |
| <i>extents</i> | The total number of disc extents that can be dynamically-allocated to the file as logical records are written to it. The size of each extent is determined by the <i>records</i> parameter value divided by the <i>extents</i> parameter value. The <i>extents</i> value must be an integer from 1 to 16. (Required parameter.) |

Designating SL's for Management by the User

To enter commands that modify an existing SL in any way, the user must first identify the SL by the `-SL` command:

`_SL filereference`

| | |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------|
| <i>filereference</i> | The name of the SL to be modified, in the <i>filereference</i> format (including group and account identifiers). (Required parameter.) |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------|

Then, all subsequent SL-modification commands will refer implicitly to this SL until a new `-SL` command is issued. If, however, the user has entered a `-BUILDSL` command, this acts as an implicit `-SL` command for the file referenced. That is, all SL-modification commands between a `-BUILDSL` command and the next `-SL` command or `-BUILDSL` command refer to the file specified in the first `-BUILDSL` command.

Adding a Procedure Segment to an SL

To add a procedure to the currently-managed SL, the user actually selects the RBM containing this procedure, and all other RBM's sharing the same segment name, from the currently-managed USL, formats these into a segment, and places this segment in the SL. The user must have write-access to the designated SL file. The procedure is added with the following command:

```
_ADDSL name[,PMAP]
```

name The name of the segment to be added to the SL. (Required parameter.)

PMAP An indication that a listing describing the prepared segment will be produced on the device specified in the :SEGMENTER command parameter *listfile*. If this parameter is omitted, the prepared segment is not listed. (Optional parameter.)

Deleting an Entry-Point or Segment From an SL

To delete an entry-point from a segment in an SL, or an entire segment from an SL, the user can enter the following command. (The segment code actually remains in the SL, but the entry-point name is withdrawn from the entry-point directory. When the last entry-point in a segment is deleted, the entire segment is automatically removed from the SL.)

```
_PURGESL [unitspec,] name
```

unitspec Is ENTRY, to delete the entry-point identified by *name*; or

 SEGMENT, to delete the segment identified by *name*. (Optional parameter.)

 The default parameter is ENTRY.

name The name of the entry-point or segment to be deleted. (Required parameter.)

Listing Procedures in an SL

To list the procedures in the currently-managed SL, the user enters:

```
_LISTSL
```

An example of the output produced by the `-LISTSL` command is shown in Figure 7-3. On this listing, significant entries are indicated with arrows, keyed to the discussion below. (All numbers appearing on this listing are octal values.)

| Item No. | Meaning |
|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1 | The name of the SL file (filename.groupname.accountname). |
| 2 | The logical segment number (relative to this SL). |
| 3 | The segment name. |
| 4 | The segment length, in words. |
| 5 | The entry-point name in the segment. |
| 6 | The parameter checking-level of the entry-point. |
| 7 | The callability of the entry point, where C = Callable U = Uncallable |
| 8 | The Segment Transfer Table (STT) number of the entry point. |
| 9 | The entry-point address (relative displacement within the segment). |
| 10 | The name of the external reference. |
| 11 | The parameter checking level of this external reference. |
| 12 | The STT number of this external reference. |
| 13 | If a number appears in this field, it is a (logical) segment number and indicates that this reference has been bound to an entry point within the SL. If a question mark appears in this field, this indicates that this reference has not been bound to any entry point within the SL. |
| 14 | A bit list of the segments referenced within the SL (with the left-most bit corresponding to the <i>0th</i> logical segment number): For each bit, 1 = Segment referenced. 0 = Segment not referenced. |
| 15 | The number of words presently used in the SL file. |
| 16 | The number of words not used, at present, in the SL file. |

Following the line containing the segment number, name, and length, any of the following segment attributes may appear. (These attributes can only be specified during system configuration. All attributes except PRIVILEGED pertain only to the system SL.)

| Attribute | Meaning |
|------------|-----------------------------------------------------|
| RESIDENT | Segment permanently resides in main memory. |
| ALLOCATED | Segment is permanently allocated in virtual memory. |
| PRIVILEGED | Segment is executed in privileged mode. |
| SYSTEM | Segment is part of MPE/3000. |

SL FILE SCR2.MPE.SYS

| SEGMENT | 0 | SEG3 | CHECK | CAL | STT | ADR |
|--------------|---|------|-------|-----|------|-----|
| ENTRY POINTS | | | | | | |
| SETRIT | 0 | | C | 20 | 1020 | |
| PARMCHECK | 0 | | C | 1 | 0 | |
| GETRECDDISP | 0 | | C | 14 | 737 | |
| FERROR | 0 | | C | 40 | 1303 | |
| REPAIRRECORD | 0 | | C | 12 | 710 | |
| TESTRIT | 0 | | C | 22 | 1040 | |
| NTOA | 0 | | C | 15 | 754 | |
| <hr/> | | | | | | |
| SUMRITS | 0 | | C | 17 | 1006 | |
| DNTOA | 0 | | C | 16 | 762 | |
| FWRITEMR | 0 | | C | 31 | 1167 | |
| MAKEROOMINDL | 0 | | C | 3 | 163 | |

| EXTERNALS | CHECK | STT | SEG |
|------------|-------|-----|-----|
| QUIT | 0 | 52 | ? |
| PRINTERROR | 0 | 51 | 1 |
| MESSAGE | 0 | 50 | 1 |
| FCHECK | 0 | 47 | ? |
| FGETINFO | 0 | 46 | ? |
| FREADDIR | 0 | 45 | ? |
| FWRITEDIR | 0 | 44 | ? |
| FWRITE | 0 | 43 | ? |
| ERROR | 0 | 42 | 1 |
| DLSIZE | 0 | 41 | ? |

110

SEGMENT 1 SEG4 LENGTH 1424

| ENTRY POINTS | CHECK | CAL | STT | ADR |
|--------------|-------|-----|-----|------|
| WARN | 0 | C | 10 | 1356 |
| ERRORN | 0 | C | 5 | 1337 |
| ERRORS | 0 | C | 4 | 1332 |
| PRINTWARNING | 0 | C | 3 | 1235 |
| DMPY | 0 | C | 11 | 1363 |
| WARNS | 0 | C | 7 | 1351 |
| MESSAGE | 0 | C | 1 | 1104 |
| PRINTERROR | 0 | C | 2 | 1210 |
| ERROR | 0 | C | 6 | 1344 |

| EXTERNALS | CHECK | STT | SEG |
|-----------|-------|-----|-----|
| CLEARLINE | 0 | 16 | 0 |
| PRINT | 0 | 15 | ? |
| DNTOA | 0 | 14 | 0 |
| SETJCW | 0 | 13 | ? |
| GETJCW | 0 | 12 | ? |

110

USED

17400

AVAILABLE

11400

Figure 7-3. -LISTSL Command Output

SETTING RBM INTERNAL FLAGS

In addition to the activity bit associated with every entry-point on an RBM, there is also an *internal flag* that determines whether or not that entry-point is to be placed in the library directory — and thus made available to users and segments within that SL — when the RBM is segmented and added to an SL. (The internal flag is only interrogated when the segment is actually added to an SL.) As an example of how this facility can be used, suppose that a programmer is writing a complicated utility routine that he wants to add to an SL. He writes this routine as a set of procedures to be placed in the same code segment, but wants only the entry-points of certain procedures made available to other users; he does not want them to use the entry-points of the support procedures for the main routine. He can effectively hide these private entry-points from users by using the OPTION INTERNAL statement (SPL/3000 only) in the declarations of the applicable procedures, or by setting the internal flag *on* with the -HIDE command described below. When these procedures are prepared onto a segment and added to an SL, the entry-points designated as internal are not entered in the directory of the library.

To set an internal flag *on*, enter:

```
-HIDE name[(index)]
```

name The name of the entry-point in the currently-managed USL whose internal flag is to be set *on*. (Required parameter.)

index The index integer further specifying the entry-point *name*; the index may be used when the RBM contains more than one entry-point of this name. If *index* is omitted, 0 is assigned by default. (Optional parameter.)

To set an internal flag to *off*, enter the following command:

```
-REVEAL name[(index)]
```

name The name of the entry-point in the currently-managed USL whose internal flag is to be set *off*. (Required parameter.)

index The index integer further specifying the entry-point *name*; the index may be used when the RBM contains more than one entry-point of this name. If *index* is omitted, a value of 0 is assigned. (Optional parameter.)

DYNAMIC LOADING OF LIBRARY PROCEDURES

Normally, segments containing library procedures referenced by a program are attached to that program when the program is allocated in virtual memory. However, the programmer may also dynamically attach and detach such procedures while his program is running. He might, for example, decide to do this for a large procedure used optionally and infrequently by his program, or for a procedure whose name is not known at load-time. By loading this procedure only when it is required, and then unloading it, he could keep the virtual memory space used by his program to a minimum most of the time. The procedures are loaded from SL libraries, not from RL libraries (which are used only at program-preparation time).

Procedures are dynamically loaded and unloaded through the intrinsic calls described below.

The user need not dynamically-load procedures declared as externals to his program, because the loader will automatically load them. Dynamic procedure loading is intended for procedures that are not declared external.

Dynamic Loading

To dynamically load a library procedure (together with external procedures referenced by it), the user issues the LOADPROC intrinsic call.

```
INTEGER PROCEDURE LOADPROC (procname,lib,plabel);  
VALUE lib;  
BYTE ARRAY procname;  
INTEGER lib,plabel;  
OPTION EXTERNAL;
```

This intrinsic returns (as the value of LOADPROC), an identity number required for use in unloading the procedure. (If an error occurs during execution of the intrinsic, an error code number rather than the identity number is returned.)

The intrinsic call parameters are

procname A byte array containing the name of the procedure to be loaded. The name must be terminated by a blank.

lib Is 2, to request library searching for the procedure in this order:

- Group Library
- Account Public Library
- System Library; or

1, to request library searching in this order:

- Account Public Library
- System Library; or

0, to request searching of the system library only.

plabel The word to which the procedure's label (P-label) is returned.

The LOADPROC intrinsic returns either of the following condition codes:

| | |
|-----|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CCE | Request granted. |
| CCG | (Not returned by this intrinsic.) |
| CCL | Request denied; the value returned to the user's process is an error-code number. (This is one of the Loader run-time errors appearing in Figure 10-5C, Section X.) |

EXAMPLE:

To dynamically load a procedure named *PROC1*, the user enters the following call. Only the system library is searched for this procedure. The label of the procedure is returned to the word *LAB*. The procedure identity number is returned to the word *PNUM*.

```
PNUM := LOADPROC (PROC1,0,LAB);
```

Dynamic Unloading

To dynamically unload a procedure, and its referenced external procedures, the user calls the following intrinsic:

```
PROCEDURE UNLOADPROC (procid);  
VALUE procid;  
LOGICAL procid;  
OPTION EXTERNAL;
```

The *procid* parameter is a word containing the procedure's identity number, returned to the user's process through the LOADPROC intrinsic.

The UNLOADPROC intrinsic returns one of the following condition codes:

| | |
|-----|---------------------------------------------------|
| CCE | Request granted. |
| CCG | (Not returned by this intrinsic.) |
| CCL | Request denied because of invalid <i>procid</i> . |

EXAMPLE

To unload the procedure that was dynamically loaded in the previous example, the user issues the following intrinsic call:

UNLOADPROC (PNUM);

SECTION VIII

Requesting Utility Operations

MPE/3000 provides commands and intrinsic calls that allow the user to request various utility operations, such as

- Listing date, time, and accounting information
- Determining job status
- Determining device status
- Obtaining devicefile information
- Transmitting messages
- Inserting comments in command stream
- Requesting ASCII/binary number conversion
- Requesting ASCII/EBCDIC (or other) character translation
- Reading Input from Job/Session Input Device
- Writing Output to Job/Session List Device
- Obtaining system timer information
- Determining the user's access mode and attributes
- Searching arrays and formatting parameters
- Executing MPE/3000 commands programmatically
- Enabling and disabling error traps
- Requesting program break, termination, or abort programmatically
- Changing the lengths of the User Managed (DL-DB) Area and Stack (Z-DL) Areas by altering DL-DB and Z-DL register off-sets
- Managing interprocess communication through the job control word
- Verifying assignment of diagnostic devices
- Changing terminal speed and echo mode

CALLS DATELINE

If the OPTION VARIABLE notation appears in an intrinsic head, some of the intrinsic parameters are optional. In this manual, these optional parameters are indicated in bold face in the intrinsic head formats.

LISTING DATE, TIME, AND ACCOUNTING INFORMATION

The current date, time, and accounting charges logged against the user's account and group can be displayed for his convenience on the job/session list device.

Date and Time

The user can print the current date and time, as recorded by the system clock, by entering:

```
:SHOWTIME
```

Accounting Charges

The user can display the total accounting information logged against his log-on account and group. This information includes usage counts and limits regarding permanent file space (in sectors), central processor time (in seconds), and on-line 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 inception of the current job/session. The user requests this display by entering:

```
:REPORT
```

NOTE: Users with System Manager or Account Manager Capability can issue a more explicit form of this command, referencing particular accounts and groups. Additionally, System Manager Users can re-set resource use counts for an account (and for all of its groups). These functions are described in MPE/3000 Operating System, System Manager/System Supervisor Manual.

EXAMPLE:

While logged-on under the account GP and the group COMLIB, the user issues a :REPORT command. In the resulting output (shown below), the specific account and group names, and counts and limits, are indicated under the corresponding headings at the top of the listing. (The double asterisks (**)) in the three LIMIT fields indicate that no limits exist.)

```

:REPORT
ACCOUNT          FILESPACE-SECTORS    CPU-SECONDS    CONNECT-MINUTES
  /GROUP          COUNT      LIMIT      COUNT      LIMIT      COUNT      LIMIT
GP                27382         **      34417         **      10637         **
  /COMLIB         2201          **           3073          **           790           **

```

DETERMINING JOB STATUS

A programmer can obtain status information about a job or session (or all currently defined jobs or sessions) with the following command:

```

:SHOWJOB [ jsnumber
          STATUS
          qualifier1 [;qualifier2]
          qualifier2 [;qualifier1] ]

```


- b. WAIT — The job/session has not been initiated due to lack of necessary resources (e.g., list device), because its input priority is less than or equal to the current job fence (deferred); or because the currently defined job limit has been attained. If the job is not deferred, the job's rank in the scheduling queue is displayed.
- c. EXEC — The job/session has been initiated and is processing. (This state does not imply that a program is currently running for the job.)
- d. SUSP — The job/session has been suspended by a =BREAKJOB command.
- e. D — for INTROduced or WAITing jobs, this indicates that the job is deferred and will not be dispatched until freed by the operator.
- f. DONE — The job is terminating.

- The job/session input device; "R" indicates it is a restartable job.
- The job/session list device or class; "S" indicates it is spooled.
- The fully-qualified job/session name.
- The day of the week and time that the job was first introduced to MPE/3000.

If no parameter is supplied to :SHOWJOB, then all jobs/sessions in the system will be displayed.

Following the display of information for jobs specified by qualifiers, the information normally produced by the *STATUS* parameter is also displayed automatically.

DETERMINING DEVICE STATUS

The following command displays information regarding a particular, or all, devices:

```
:SHOWDEV { ldev
           classname }
```

ldev A specification for display of the information for the designated logical device.

classname A specification for display of the information for all devices in the specified class.

If no parameter is included with the :SHOWDEV command, the information is displayed for all devices.

The information displayed for each device includes:

Logical device number.

J — Job accepting.

D — :DATA accepting

A — Job and :DATA accepting

Availability

AVAIL — available as a real, non-sharable device.

SPOOLED — available via input or output spooling.

UNAVAIL — unavailable (owned).

DOWN.

DISC.

Ownership (if applicable) of the device:

SYS — Owned by the system; if “#nnn” appears, it specifies the owning PIN.

SPOOLER — (IN or OUT).

#Jnnn or

#Snnn — Owned by the indicated job.

DIAG. — =GIVEN (allocated) to diagnostics.

nnn FILES — Indicates the number of files currently opened on the device.

DP — Indicates that DOWN is pending.

OBTAINING DEVICEFILE INFORMATION

In addition to obtaining device status information (via the preceding command), the user may also obtain information about any or all devicefiles. The :SHOWIN command displays information about input devicefiles, and the :SHOWOUT command displays information about output devicefiles. The :SHOWIN command is written as follows:

```
:SHOWIN [ #Innn  
          STATUS  
          qualifier [;qualifier] . . . ]
```

#Innn A request for a particular file.

STATUS A request for summarizing information regarding the devicefiles currently existing for the user's job.

qualifier

Specifies a list of input devicefiles which satisfy the following qualifications:

$$\left[\begin{array}{l} DEV=ldev \\ JOB= \left\{ \begin{array}{l} @J \\ @S \\ jsnum \end{array} \right\} \\ \begin{array}{l} ACTIVE \\ READY \\ OPENED \end{array} \end{array} \right]$$

where: *ldev* is a logical device number; @J requests jobs only; @S requests sessions only; *jsnum* requests a particular job or session number; *ACTIVE*, *READY*, and *OPENED* are specific devicefile states.

If no parameter is given, all input devicefiles are displayed.

This command displays only input devicefile information, which includes:

- *devicefileid*, in the form #Innn;
- state;
- assigned device;
- *jobnum*, if not *READY* or *ACTIVE* :DATA; otherwise, the job name appears on a line following the standard device information.
- *filename*;
- forms message indication (but not the message);
- for spooled input devicefiles, the amount of disc space currently used.

The :SHOWOUT command is written as follows:

$$:SHOWOUT \left[\begin{array}{l} \#Onnn \\ STATUS \\ qualifier [;qualifier] \dots \end{array} \right]$$

#Onnn A request for a particular file.

STATUS A request for summarizing information regarding the output devicefiles currently existing for the user's job.

qualifier

Specifies a list of output devicefiles which satisfy the following qualifications.

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

where: *ldev* is a logical device number; *classname* is a specific device class; *@J* requests jobs only; *@S* requests sessions only; *jsnum* requests a particular job or session number; *ACTIVE*, *READY*, and *OPENED* are specific devicefile states; *N* requests only nondeferred files, and *D* requests only deferred files.

If no parameter is given, all output devicefiles are displayed.

This command displays only output devicefile information, which includes:

- *devicefileid*, in the form #*Onnn*;
- state;
- assigned device or output class;
- *jobnum*, if not *READY* or *ACTIVE* :*DATA*; otherwise, the job name appears on a line following the standard device information.
- *filename*;
- forms message indication (but not the message);
- for spooled output devicefiles:
 - a. outputpriority;
 - b. number of copies;
 - c. deferred indication, "D"; or "rank" in the output scheduling queue;
 - d. amount of disc space currently used.

TRANSMITTING MESSAGES

Any user can transmit messages from his job or session to other jobs or sessions currently running, by entering the `:TELL` command. The message appears on the receiving user's standard list device. If a terminal to which a message is sent is currently receiving program output, this message is queued as high as possible among the current input/output requests, but does not interrupt reading or writing in progress. The format of the `:TELL` command is

$$:TELL \left\{ \begin{array}{l} j\text{number} \\ j\text{name} \end{array} \right\}; \text{message}$$

jnumber The job or session number assigned by MPE/3000 to the job/session that is to receive the message. It is entered in this format:

#Jnnnnn or #Snnnnn

jname The name of the job or session that is to receive the message. It is written as entered in the `:JOB` command, or the `:HELLO` command, in the format:

[jname,] username.acctname

If several jobs or sessions coincidentally have the same name, only one of these will be selected (arbitrarily); in such a case, the user should use the *jnumber* parameter with the `:TELL` command.

message The message, comprised of any string of ASCII characters. The message is terminated by the end of the record on which it appears, or by a carriage return. (Required parameter.)

The message appears on the receiving list device in this format:

FROM/[jname,] username .acctname/message

jname } The names of the transmitting job/session and user,
username } and the name of the account under which they are
acctname } running.

message The message, in ASCII characters.

NOTE: *The `:TELL` command is rejected if the final destination message (FROM/ . . .) exceeds 72 characters. It is also rejected for Quiet Mode jobs and sessions (see `:SETMSG` command).*

If the job/session or user designated to receive the message is not running, the transmitting job/session will receive a message indicating this.

EXAMPLE

A user identified as SMITH, running a session named SMITHSES, wants to send a message to a user identified as BROWN, running a session named BROWNSES. (Both users are logged-on under account A.) SMITH enters:

```
:TELL BROWNSES,BROWN.A;SYSTEM WILL BE SHUT DOWN AT 4:30 TODAY.
```

BROWN receives:

```
FROM/SMITHSES,SMITH.A/SYSTEM WILL BE SHUT DOWN AT 4:30 TODAY.
```

The user can send a message to the MPE/3000 operator's console by entering the `:TELLOP` command:

`:TELLOP message`

message The message to be transmitted. The *message* is limited to approximately 57 characters.

The user has the option to run his job or session in Quiet Mode, wherein `:TELL` messages from other users and `=TELL` messages from the console operator will be refused. (However, `=WARN` messages from the console operator can override the Quiet Mode condition.) The `:SETMSG` command is used to enable or disable this mode:

```
:SETMSG { ON }  
         { OFF }
```

OFF Sets the job or session in such a state that messages from other users (and `=TELL` messages from the console operator) will be refused; i.e., runs in Quiet Mode.

ON Re-enables the reception of messages.

INSERTING COMMENTS IN COMMAND STREAM

Comments may be included in the MPE/3000 command stream by using the following command:

```
:COMMENT text
```

Like all other user commands, lines of the :COMMENT command are assumed to be continued when the last non-blank character on the line is an ampersand.

REQUESTING ASCII/BINARY NUMBER CONVERSION

The user can convert decimal or octal numbers represented in ASCII code to their binary equivalents, and vice-versa, with the intrinsics described below.

Converting Numbers from ASCII to Binary Code

To convert an ASCII-coded number to its binary equivalent, the user issues the BINARY intrinsic call.

LOGICAL PROCEDURE

BINARY (string, length);

VALUE length;

BYTE ARRAY string;

INTEGER length;

OPTION EXTERNAL;

This intrinsic returns the converted value to the user's program (as the value of BINARY). In this intrinsic call, the parameters are

- string* A byte array containing the signed octal or decimal number (in ASCII code) to be converted. If the character string in this array begins with a percent sign (%), it is treated as an octal value; if the string begins with a plus sign, minus sign, or digit, it is treated as a decimal value.
- length* An integer representing the length (number of bytes) in the byte array containing the ASCII-coded value. If the value of *length* is 0, the intrinsic returns 0 to the calling process. If the value of *length* is less than 0, the intrinsic aborts.

The execution of the BINARY intrinsic can result in one of these three condition codes:

- CCE Successful conversion; a one-word binary value is returned to the user's program.
- CCG A word overflow, possibly resulting from too many characters, occurred in the word returned.
- CCL An illegal character was encountered in the byte array specified by *string*. This could include the digits 8 or 9 specified in an octal value.

EXAMPLE:

Suppose the byte array *STRINGA* contained the ASCII characters "%16". The user wants to convert this string to its binary equivalent and store the result in the word *RESULT*. He issues the following call:

RESULT := BINARY (STRINGA,3);

The binary value 1110 (decimal 14) is stored in *RESULT*.

The user can request double-integer ASCII-to-binary code conversion by calling the *DBINARY* intrinsic. Except for a double-word result, this intrinsic is identical to the *BINARY* intrinsic just described.

DOUBLE PROCEDURE

DBINARY (string, length);

VALUE length;

BYTE ARRAY string;

INTEGER length;

OPTION EXTERNAL;

This intrinsic returns (as the value of *DBINARY*) the converted double-word value. The parameters and possible condition codes are the same as those of the *BINARY* intrinsic.

Converting Numbers from Binary to ASCII Code

The user can convert any 16-bit binary number to its ASCII equivalent by entering the *ASCII* intrinsic call.

INTEGER PROCEDURE

ASCII (word, base, string);

VALUE word, base;

LOGICAL word;

INTEGER base;

BYTE ARRAY string;

OPTIONAL EXTERNAL;

This intrinsic returns to the caller (as the value of ASCII) the number of characters in the final conversion. The parameters are

word The number to be converted to ASCII code.

base An integer indicating octal or decimal conversion.

 8 = octal
 10 = decimal (left justified)
 -10 = decimal (right justified)

If any other number is entered in this parameter, the intrinsic aborts.

string The byte array into which the converted value is to be placed. This array must be long enough to contain the result. (No result, however, exceeds six characters.)

For octal conversion (base = 8), six characters (including leading zeros) are always returned in *string*, showing the octal representation of *word*. For octal conversions, the result of ASCII is the number of significant (right-justified) characters in *string*. (If *word* = 0, the result of the intrinsic is 1.)

For decimal conversions (base = 10), *word* is considered as a 16-bit, 2's-complement integer (ranging from -32768 to +32767). Leading zeros are removed and the ASCII result is left-justified. In decimal conversions, if the value is negative, the first byte of *string* contains a minus sign. If *word* contains a single zero, only 0 is returned to *string*. For decimal conversions, the result of ASCII is the total number of characters in *string* (including the sign). (If *word* equals 0, the result of ASCII is 1.)

For decimal left-justified conversions (base = 10), *word* is considered as a 16-bit, 2's-complement integer (ranging from -32768 to +32767). Leading zeros are removed and the ASCII result is left-justified. In decimal conversions, if the value is negative, the first byte of *string* contains a minus sign. If *word* contains a single zero, only 0 is returned to *string*. For decimal conversions, the result of ASCII is the total number of characters in *string* (including the sign). (If *word* equals 0, the result of ASCII is 1.)

For decimal right-justified conversions (base = -10), *word* is considered as a 16-bit 2's-complement integer and converted into an ASCII decimal representation. The result is right-justified with *string* defining the *right-most* byte in the field of the result. The intrinsic returns the number of ASCII characters in the conversion.

With this intrinsic, the condition code remains unchanged.

EXAMPLE:

Suppose the word *BWORD* contained %177666. The user wants to convert this value to its decimal equivalent, returning it to the byte-array *RSTRING* and returning the number of characters in the conversion to the word *LEN*. He issues this call:

```
LEN := ASCII (BWORD,10,RSTRING);
```

Then, *RSTRING* contains "-74" and *LEN* contains 3.

The user can convert a 32-bit (double-word) binary number to its ASCII equivalent by entering the DASCII intrinsic call.

```
INTEGRAL PROCEDURE      DASCII (dword, base, string);
VALUE dword, base;
DOUBLE dword;
INTEGER base;
BYTE ARRAY string;
OPTION EXTERNAL;
```

The DASCII intrinsic returns to the caller the number of characters in the final conversion. The parameters are

dword A double-word value indicating the number to be converted to ASCII code.

base } The same as the corresponding parameters for the ASCII intrinsic, except
string } that *string* always receives 11 characters for octal conversions, and may
 receive up to 11 characters for decimal conversions; a negative *base* (for right
 justification) is not valid for DASCII.

With this intrinsic, the condition code remains unchanged.

REQUESTING CHARACTER TRANSLATION

The user can convert a string of characters represented in EBCDIC code to its ASCII equivalent, and vice versa, with the CTRANSLATE intrinsic. He can also use CTRANSLATE to translate to other codes by defining his own translation table. The intrinsic call for CTRANSLATE:

```
PROCEDURE      CTRANSLATE (code, instring, outstring, stringlength, table);
VALUE          code, stringlength;
INTEGER        code, stringlength;
BYTE ARRAY    instring, outstring, table;
OPTION VARIABLE, EXTERNAL;
```

In this intrinsic call, the parameters are:

code An integer identifying a specific translation to be used.

- 0 Use the user-supplied translation table given by the parameter *table*.
- 1 Use the EBCDIC-to-ASCII table. Those EBCDIC characters which have no ASCII equivalent will translate into a byte of zero.
- 2 Use the ASCII-to-EBCDIC table. The ASCII parity bit is ignored.

instring The string of characters to be translated.

outstring An optional parameter specifying a byte array to which is returned the translated character string. If *outstring* is not specified, all translation will occur within *instring*. The parameters *instring* and *outstring* may specify the same array.



stringlength A positive integer specifying the length (in bytes) of *instring*.

table A byte array whose ordering and contents define the translation process. This parameter is required only when *code* = 0. This table is constructed such that, for each byte, a table entry is a byte in the target character set and is located at a byte position whose displacement from *table* is the numeric value of the corresponding character in the source character set. The length of *table* may be as large as 256 bytes, but it needs to be only as large as the largest numeric value of any source byte in *instring*.

The execution of the CTRANSLATE intrinsic can result in one of these two condition codes:

CCE Request granted; translation successfully performed.
 CCL Request denied because an error occurred.

EXAMPLE

Suppose the byte array *ESTRING* contains the EBCDIC characters "JOB 2". The user wants to convert this string to its ASCII equivalent and store the result in the byte array *ASTRING*. He issues the following call:

```
CTRANSLATE (1,ESTRING,ASTRING,5);
```

The byte array *ASTRING* will then contain the ASCII characters for "JOB 2".

TRANSMITTING PROGRAM INPUT/OUTPUT (FROM JOB/SESSION INPUT/LIST DEVICES)

In addition to the FREAD and FWRITE intrinsics discussed in Section VI, MPE/3000 provides three other intrinsics that permit the user to read information from the job/session input device or write information to the job/session listing device. (The user does not issue FOPEN against these devices prior to issuing these intrinsics.) These commands are primarily used by programmers designing their own subsystems or command executors, to read commands or write error messages. Other programmers typically use the file system commands for general input, and listing output.

Reading Input

The job/session input device acts as the source of all MPE/3000 commands relating to a job or session, and the primary source of all ASCII information input to the job or session. This is normally a card reader for jobs and a terminal for sessions.

The programmer can read a string of ASCII characters from a job/session input device into an array in his program by calling the READ (or READX) intrinsic. (This intrinsic call is equivalent to issuing an FREAD call against the file \$STDIN.) Basic line editing such as cancellation of lines and backspacing are performed automatically by the input/output driver. If the input device is a terminal and it is in full-duplex mode and the echo facility is on, or if the terminal is in half-duplex mode, the characters read are printed.

INTEGER PROCEDURE

READ (*message*,*expectedl*);

VALUE *expectedl*;

ARRAY *message*;

INTEGER *expectedl*;

OPTION EXTERNAL;

The READ intrinsic returns to the caller the positive length of the input actually read, in bytes (if *expectedl* is negative) or words (if *expectedl* is positive), as the value of READ. The parameters for this intrinsic call are

message The array into which the ASCII characters are read.

expectedl An integer denoting the maximum length of the array *message*. If *expectedl* is negative, this specifies the length in bytes; if *expectedl* is positive, it indicates the length in words. When the record is read, the first *expectedl* characters are input. If *expectedl* equals or exceeds the size of the physical record, the entire record is transmitted.

Device mode settings specified through the FCONTROL or FSETMODE intrinsics on the file \$STDIN appropriately affect transactions using the READ intrinsic.

The READ command can result in the following condition codes:

CCE The record was read into the area specified by *message*.

CCG A record with a colon in the first column was encountered, signalling the end of data.

CCL A physical input/output error occurred, and *message* is not valid; further error analysis, through the FCHECK intrinsic, is not possible.

EXAMPLE:

To read information from a record input through the job input device into an array named INBUFF whose size is 72 bytes, the user issues the following call:

```
LEN := READ (INBUFF,-72);
```

LEN will contain the (positive) number of characters actually transmitted.

INTEGER PROCEDURE

READX (message,expectedl);

VALUE expectedl;

ARRAY message;

INTEGER expectedl;

OPTION EXTERNAL;

The READX intrinsic is identical to READ except that it reads from \$STDINX. The CCG condition code will be returned only when :EOD, :EOJ, :JOB, or :DATA is encountered.

Writing Output to the Listing Device

All MPE/3000 commands issued and any system messages for the user are copied to the job/session listing device; this device is the primary destination of all ASCII output. Normally, this device is a line printer for jobs and a terminal for sessions.

The programmer can write a string of ASCII characters from an array in his program to the job/session listing device by calling the PRINT intrinsic. This intrinsic call is equivalent to issuing a FWRITE call against the file \$STDLIST.

PROCEDURE

PRINT (message,length,control);

VALUE length,control;

ARRAY message;

INTEGER length,control;

OPTION EXTERNAL;

In this intrinsic call, the parameters are

message The array from which the characters are output.

length An integer denoting the length of the output string to be written. If *length* is *negative*, this specifies the length in bytes; if it is positive, it indicates the length in *words*.

control An integer representing a carriage control code (as described under the discussion of the FWRITE intrinsic in Section VI).

Device control settings specified through the FCONTROL or FSETMODE intrinsics on the file \$STDLIST appropriately affect transactions using the PRINT intrinsic.

A secondary entry-point to the PRINT intrinsic, PRINT', coincides with the main entry point and is provided primarily for use in compilers that require a reserved intrinsic inaccessible to their users.

The condition codes returned are

| | |
|-----|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CCE | The intrinsic was successfully executed. |
| CCG | End-of-data was encountered. |
| CCL | Physical input/output error. The character string specified was not successfully written. Further error analysis, through the FCHECK intrinsic, is not possible. |

EXAMPLE:

The following intrinsic call prints the number of characters specified by XCHARS from the array labeled OUTBUFF onto the job listing device. The first character of the string is used for carriage control.

```
PRINT (OUTBUFF,XCHARS,1)
```

Writing Output to the Operator's Console

The programmer can also transmit information from an array to the operator's console. He does this with the PRINTOP intrinsic call:

```
PROCEDURE PRINTOP (message,length,control);  
VALUE length, control  
ARRAY message;  
INTEGER length, control;  
OPTION EXTERNAL;
```

The parameters and condition codes for this intrinsic are identical to those for the PRINT intrinsic, except that *message* is limited to 56 characters.

Also, the programmer can transmit information from an array to the operator's console and solicit a reply which is returned to the user for examination. He does this with the PRINTOPREPLY intrinsic call:

```
INTEGER PROCEDURE PRINTOPREPLY (message,length,control,reply,expectedl);  
VALUE length, control, expectedl;  
ARRAY message, reply;  
INTEGER length, control, expectedl;  
OPTION EXTERNAL;
```

| | |
|------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>message</i> | The array from which the characters are output to the operator's console. |
| <i>length</i> | An integer denoting the length of the output string to be written. If <i>length</i> is negative, it denotes a byte count; if it is positive, it denotes a word count. This parameter should never specify a message length of more than 50 bytes. The length of the output differs from that generated by PRINTOP because the process identification number and length of reply are also supplied. |
| <i>control</i> | An integer representing a carriage control code. |
| <i>reply</i> | The array into which input characters are read from the operator's console. |
| <i>expectedl</i> | An integer denoting the maximum length of the message to be read into the array <i>reply</i> , if <i>expectedl</i> is negative, it denotes a byte count; if it is positive, it denotes a word count. This parameter should never specify a reply length of more than 31 bytes. |

The PRINTOPREPLY intrinsic returns to the caller a positive integer indicating the actual length of the input reply from the console operator. This length reflects a byte count if *expectedl* is negative or a word count if *expectedl* is positive.

If *expectedl* is zero, then the PRINTOPREPLY intrinsic behaves like PRINTOP and does not solicit a reply. In this case, the intrinsic value returned by PRINTOPREPLY will be zero.

When CCL is returned by PRINTOPREPLY, the intrinsic value will be zero.

The parameter *length* may be zero, in which case only the standard message prefix is written on the operator's console. If both *length* and *expectedl* are zero, then the intrinsic returns a CCL.

Condition codes:

| | | |
|-----|---|-----------------------------------------|
| CCE | = | the intrinsic was successfully executed |
| CCG | = | (cannot occur) |
| CCL | = | error occurred |

OBTAINING SYSTEM TIMER INFORMATION

The programmer can request the return of system timer information to his program with the intrinsic calls described below.

System Timer Bit-Count

The user can return to his program a 31-bit logical quantity representing the current system timer and timer overflow count. This count can be used in routines that generate random numbers, or in measuring the real-time (wall-clock time) elapsed between two events. The resolution of the system timer is one millisecond; in other words, readings taken within a one-millisecond period may be identical. (The user should bear in mind that this timer-count

is a hardware representation that is meaningless except for its role in random-number initialization.) The count is obtained with this intrinsic:

```
DOUBLE PROCEDURE TIMER;
OPTION EXTERNAL;
```

The timer count is returned as the value of `TIMER`. The condition code is not changed by this intrinsic.

Current Time

The user can request that the current date and time, as monitored by the system timer, be returned to his process as a triple word. The first word contains the year (last two digits) and day-of-the-year; the second word contains the hour (24-hour clock) and minute; the third word contains the second and tenth-of-a-second. The date and time are requested with the `CHRONOS` intrinsic.

```
LONG PROCEDURE CHRONOS;
OPTION EXTERNAL;
```

The format of the returned information is

| | | |
|--------------------------|-----------------|----|
| 0 | 6/7 | 15 |
| Last 2 digits of year | Day of year | |
| 0:7 | 7:9 | |
| Hour | Minute | |
| 0:8 | 8:8 | |
| Second | Tenth-of-Second | |
| 0:8 | 8:8 | |

CALENDAR
CLOCK

The triple word is returned as the value of `CHRONOS`. The condition code is not changed by this intrinsic.

The user can also request either the date or the time, individually, by either the `CALENDAR` or the `CLOCK` intrinsic calls. The returned values are subsets of the triple word that is returned for `CHRONOS`. `CALENDAR` returns the date and year in one word having the same format as the first word in the `CHRONOS` format, above. `CLOCK` returns the time in a double word having the same format as the last two words in the `CHRONOS` format. The calls for these intrinsics are:

```
LOGICAL PROCEDURE CALENDAR;
OPTION EXTERNAL;

DOUBLE PROCEDURE CLOCK;
OPTION EXTERNAL;
```

The condition code is not changed by these intrinsics.

24
24 5/4
1 4 3/4
62

REPLACES

The user can also cause the calling process to be suspended for a specified number of seconds by calling the PAUSE intrinsic.

PROCEDURE

PAUSE (*interval*);

REAL *interval*;

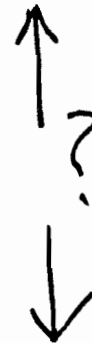
OPTION EXTERNAL;

In this intrinsic, *interval* is a positive real value specifying the amount of time, in milliseconds, that the process will pause (required parameter). The maximum time allowed is approximately 2000 seconds.

The PAUSE intrinsic call is awakened automatically, when at least *interval* seconds have passed, and control is returned to the caller.

The condition codes returned are

- | | |
|-----|------------------------------------------------------------------------------------------------------|
| CCE | Request granted. |
| CCG | Not returned by this intrinsic. |
| CCL | Request denied because a negative value was specified for <i>interval</i> or the value is too large. |





DETERMINING THE USER'S ACCESS MODE AND ATTRIBUTES

A program can obtain the access mode and attributes of the user running that program from the system tables describing the user. This information is requested with the WHO intrinsic call:

PROCEDURE

| |
|--------------------------------------------------------------------|
| <i>WHO (mode,capability,lattr,usern,groupn,acctn,homen,termn);</i> |
|--------------------------------------------------------------------|

LOGICAL mode,termn;

DOUBLE capability, lattr;

BYTE ARRAY usern,groupn,acctn,homen;

OPTION VARIABLE,EXTERNAL;

The parameters in this intrinsic call are as follows. All are optional parameters specifying locations to which information is to be returned. If any parameter is omitted, the corresponding information is not returned, by default.

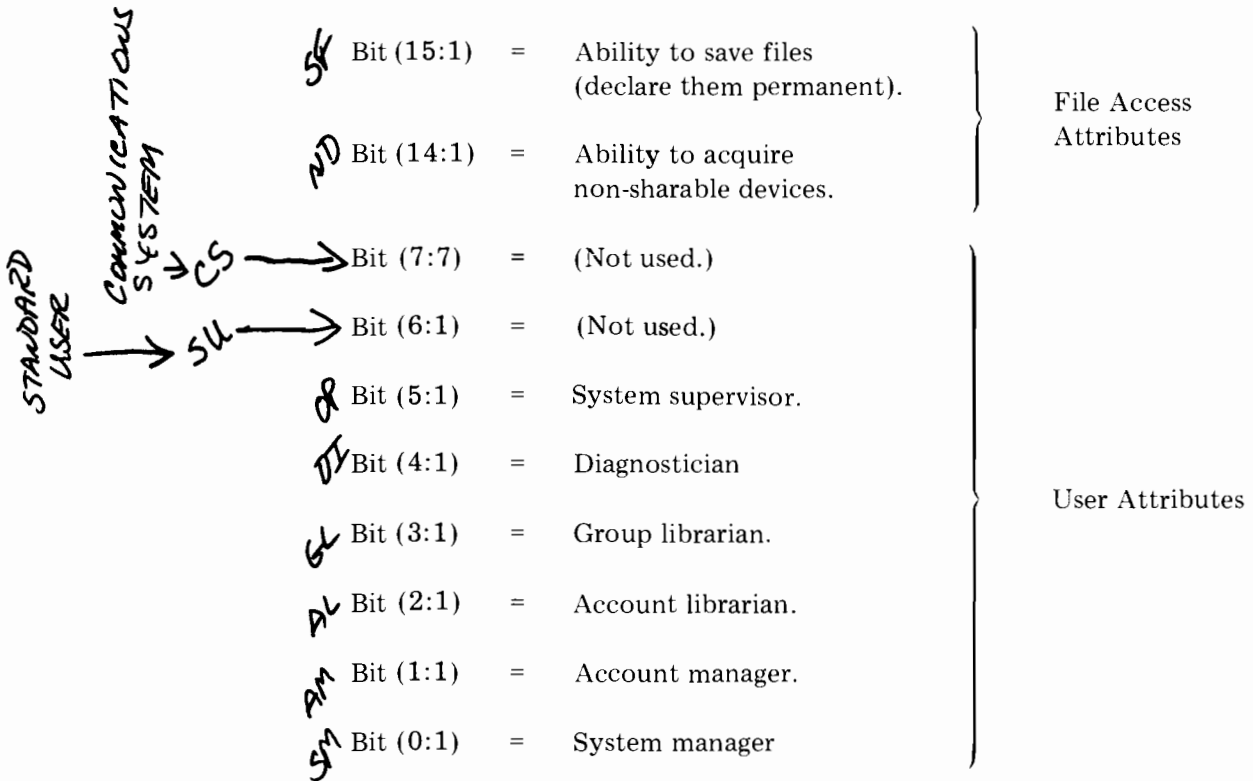
mode

A word to which the current user's mode of access will be returned. In this word, the bits will have the following meaning. (Bit positions are numbered from left to right, 0 through 15.)

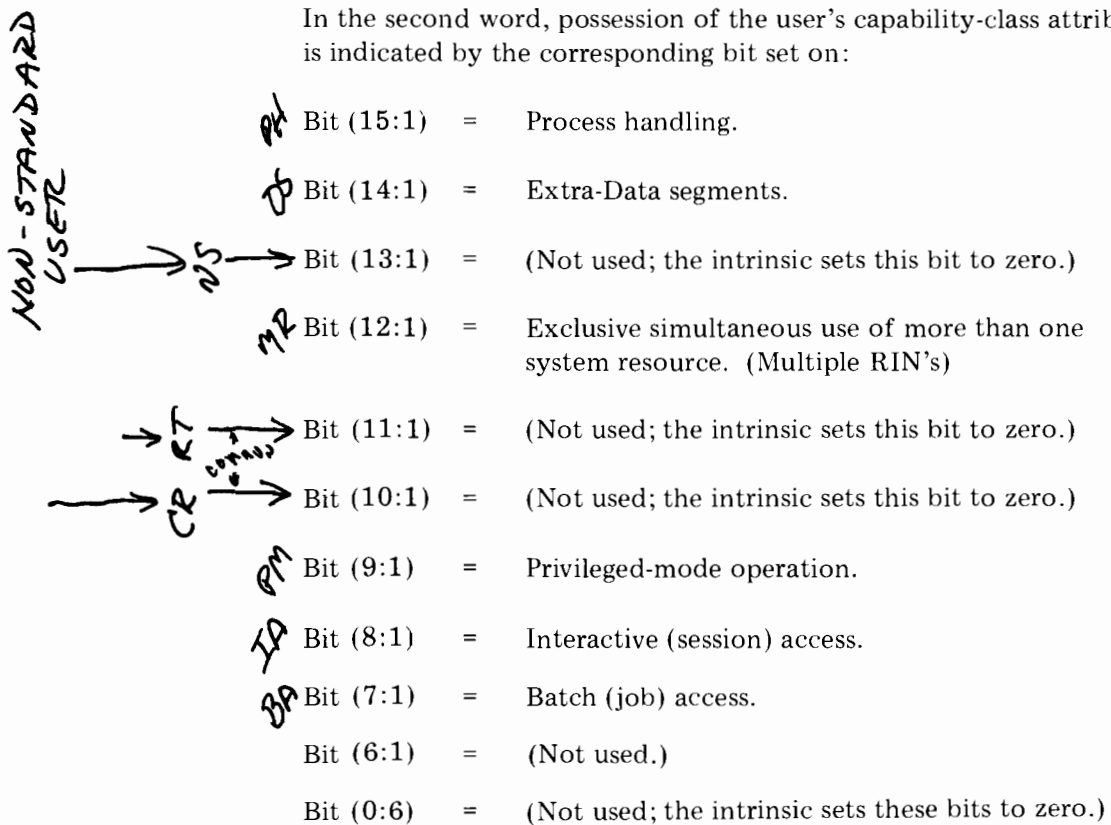
| Bits | Access Mode |
|--------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| (15:1) | <p>1 = The job/session input file and job/session list file form an <i>interactive</i> pair. (A dialogue can be established between a program (displaying information on the job list device) and a person (responding through the job input device).)</p> <p>0 = The job/session input file and job/session list file are <i>not</i> interactive.</p> |
| (14:1) | <p>1 = The job/session input file and job/session list file form a <i>duplicative</i> pair. (Images on the job input device are automatically duplicated on the job list device.)</p> <p>0 = The job/session input file and job/session list file are <i>not</i> duplicative.</p> |
| (12:2) | <p>01 = The user is accessing the system through a <i>session</i>.</p> <p>10 = The user is accessing the system through a <i>job</i>.</p> |
| (0:12) | (These bits are not used; the intrinsic always sets them to zero.) |

capability

A double-word where the user's file access, user, and capability class attributes will be returned. In the first word, possession of the following file-access and user attributes is indicated by the corresponding bit set on (equal to 1).



In the second word, possession of the user's capability-class attributes is indicated by the corresponding bit set on:



| | |
|---------------|----------------------------------------------------------------------------------------------------------------------------------|
| lattr | A double-word to which is returned the <i>local attributes</i> of the user, as defined by a user with Account Manager attribute. |
| usern | An 8-byte array where the user's name is returned. |
| groupn | An 8-byte array where the name of the user's log-on group is returned. |
| acctn | An 8-byte array where the name of the user's log-on account is returned. |
| homen | An 8-byte array where the name of the user's home group is returned. |
| termn | A word where the logical device number of the job/session input device is returned. |

The names in *usern*, *groupn*, *acctn*, and *homen* are returned left-justified, padded at the right with blanks.

The condition code is not changed by this intrinsic.



EXAMPLE

Suppose the programmer wants to return to his program the attributes assigned to the user by the system. He wants this information stored in the areas noted:

| Characteristic | Location Where Stored |
|-------------------------|-----------------------|
| <i>Mode</i> | <i>AMODE</i> |
| <i>Capability</i> | <i>CAPLIST</i> |
| <i>User Name</i> | <i>UNAME</i> |
| <i>Account Name</i> | <i>ANAME</i> |
| <i>Terminal Address</i> | <i>TADDR</i> |

He can accomplish this by entering:

```
WHO (ADMODE,CAPLIST,,UNAME,,ANAME,,TADDR);
```

OBTAINING PROCESS RUN-TIME (USE OF THE CENTRAL PROCESSOR)

The user can obtain a double integer showing the duration, in milliseconds, that the process has been running. He does this by issuing the PROCTIME intrinsic call.

```
DOUBLE PROCEDURE PROCTIME;  
  
OPTION EXTERNAL;
```

The process run-time is returned as the value of PROCTIME. The condition code is not changed by this intrinsic.

SEARCHING ARRAYS

Occasionally, a user constructs byte arrays whose contents he may later want to search for specified entries or names. A dictionary of commands designed by the user is one such example. The SEARCH intrinsic can assist the user with specially-formatted arrays consisting of sequential entries, each including:

- One byte specifying the length (in bytes) of the entire entry — the length includes this byte plus all the information in the following byte areas
- One byte specifying the *length* of the “name” (in bytes) for which the search is performed
- A byte string forming the name for which the search is performed — in a command dictionary, this is the command name
- An optional byte string containing a definition — in a command dictionary, this would be command-relevant information

The entry-number of the first entry in such a dictionary is *one*. The last entry is indicated by an entry-length of *zero*.

The user can request the search of such an array for a specified name with the SEARCH intrinsic call. A simple linear search is then performed, with the name (specified as a byte-array by the user) compared against the byte-array forming the name in each entry. (Because the search is linear, the most frequently-used name byte-arrays should appear at the beginning of the array to promote efficient searching.) If the name is found, the number of the entry containing the name is returned to the user’s program. If the name is not found, a zero is returned. At his option, the user can also request the return of a pointer to the definition information for the name.

The search is requested with the SEARCH intrinsic call:

```
INTEGER PROCEDURE SEARCH (target,length,dict,defn);  
VALUE length;  
BYTE ARRAY target,dict;  
INTEGER length;  
BYTE POINTER defn;  
OPTION VARIABLE,EXTERNAL;
```

This intrinsic returns (as the value of SEARCH) the entry number of the definition. In this intrinsic call, the parameters are

- target* The byte array containing the name for which the search is performed.
- length* An integer specifying the length, in bytes, of the byte-array *target*.
- dict* The specially-formatted byte-array in which *target* is sought.
- defn* A word to which is returned the relative byte address of the definition portion of the entry sought in the array. If omitted, this address is not returned.

The intrinsic searches *dict* for a word matching *target*, and returns the corresponding entry number to the user's process. If the matching word is not found, an entry number of zero is returned.

The condition code is not changed by this intrinsic.

EXAMPLE:

Consider a byte array, wherein the relationship of each entry to its user-relevant definition is:

| Entry | User-Relevant Definition |
|---------|--------------------------|
| THIS | 200 |
| IS | |
| AN | "XYZ" |
| EXAMPLE | "X" |

The byte-array itself, named *DEFF*, is structured as follows:

| | |
|-----|---|
| 7 | 4 |
| T | H |
| I | S |
| 200 | 4 |
| 2 | I |
| S | 7 |
| 2 | A |
| N | X |
| Y | Z |
| 10 | 7 |
| E | X |
| A | M |
| P | L |
| E | X |
| 0 | |

Suppose that the user wants to search the byte-array *DEFF* for the byte-array *NAME* (containing "AN"). Suppose, also, that *NLENGTH* is the length of the array *NAME* (2 bytes). The byte-address of the definition sought is to be returned to the word *DEFADR*. The entry-number corresponding to the definition is to be returned (as the value of *SEARCH*) to the word *ENUM*. The user issues the following call:

ENUM := SEARCH (NAME,NLENGTH,DEFF,DEFADR);

When this intrinsic is executed, *ENUM* contains 3 and *DEFADR* contains the byte pointer to "XYZ."

FORMATTING COMMAND PARAMETERS

The programmer can, within his program, extract and format for execution the parameters of a command (that is *not* an MPE/3000 command) through the MYCOMMAND intrinsic call. This intrinsic also allows the programmer, at his option, to request searching of a byte array (serving as a command dictionary) for a specified command.

INTEGER PROCEDURE

| |
|---------------------------------------------------------------------------------------------------------------------------------------------|
| <i>MYCOMMAND</i> (<i>comimage</i> , <i>delimiters</i> , <i>maxparms</i> , <i>numparms</i> , <i>parms</i> , <i>dict</i> , <i>defn</i>); |
|---------------------------------------------------------------------------------------------------------------------------------------------|

VALUE maxparms;

BYTE ARRAY comimage,delimiters,dict;

INTEGER maxparms,numparms;

DOUBLE ARRAY parms;

BYTE POINTER defn;

OPTION VARIABLE,EXTERNAL;

If *dict* is specified, the command entry number is returned as the value of MYCOMMAND. In this call, the following parameters apply:

comimage

A byte array that contains either:

- A command name (expected if the *dict* parameter is specified), followed by parameters, followed by a carriage return. The command name is delimited by the first non-alphabetic character, and cannot be preceded by any leading blanks. The parameters will be formatted and the byte array specified by *dict* will be searched for a name matching the command.
- Only command parameters (expected if the *dict* parameter is not specified), followed by a carriage return. These parameters will be formatted.

In the byte array named for the *comimage* parameter, the first character of the command or parameter list may be a leading blank.

delimiters

A word containing a string of up to 32 legal delimiters (each of which is an ASCII special byte). The last of these must be a carriage return. Each delimiter is later identified by its position in this string. If this parameter is omitted, the delimiter array "comma, equal, semicolon, carriage-return" is used.

maxparms An integer specifying the maximum number of parameters expected in *comimage*.

numparms A word to which is returned the number of parameters actually found in *comimage*.

parms A double array of *maxparms* double words that, on return, delineates the parameters. When the intrinsic is executed, the first *numparms* double words are returned to the user's process in this array, with the first double word corresponding to the first parameter, the second double word corresponding to the next parameter, and so forth. The parameter fields of *comimage* are delimited by the delimiters specified in *delimiters*. In formatting, leading and trailing blanks around each delimiter are removed and lower-case letters are upshifted. Each double word in the array named by *parms* contains the following information:

Word 1: Contains the byte pointer to the first character of the parameter.

Word 2: Contains bits that describe the parameter:

Bit (11:5) denotes the delimiter number in *delimiters* (starting at zero).

Bit (10:1) if *on*, indicates that the parameter contains special characters.

Bit (9:1) if *on*, indicates that the parameter contains numeric characters.

Bit (8:1) if *on*, indicates that the parameter contains alphabetic characters.

Bit (0:8) indicates the length of the parameter, in bytes. This value is zero if the parameter is omitted.

dict A byte-array that will be searched for the command name in *comimage*. The format must be identical to that of the *dict* parameter in SEARCH. (Actually, the command, delimited by a blank, is extracted from *comimage*, and the SEARCH intrinsic is called with the command used as the *target* parameter in SEARCH.) If the command is found in *dict*, its entry number is returned to the user's program. If the command is *not* found, or if the *dict* parameter is not specified, *zero* is returned. If *dict* is specified but the command is not found in *dict*, the parameters specified in *comimage* are *not* formatted.

defn A word to which is returned the relative address of the definition portion of the command entry in *dict*.

The MYCOMMAND intrinsic call can result in the following condition code settings:

- CCE The parameters were formatted, without exception. If *dict* was specified, the command entry number was returned to the user's program.
- CCG More parameters were found in *comimage* than were allowed by *maxparms*. Only the first *maxparms* of these parameters were formatted in *parms* and returned to the user.
- CCL The *dict* parameter was specified, but the command name was not located in the array *dict*. The parameters in *comimage* were not formatted.

The MYCOMMAND intrinsic aborts if the number of characters between delimiters exceeds 255.

EXAMPLE:

The user wants to parse PARMARRAY, allowing comma, semicolon, period, and carriage-return as delimiters. These delimiters are found in the array DELS, as follows:

| | |
|---|------|
| , | ; |
| . | (CR) |

The user wants to specify that no more than 10 parameters are expected in PARMARRAY, and that the number of parameters actually found in PARMARRAY be returned to NUM. The delineation information for the parameters is to be returned to the double-array PARMS. He issues the following call:

MYCOMMAND (PARMARRY, DELS, 10, NUM, PARMS);

Suppose PARMARRAY starts at DB-relative Address %1200 and contains the following ASCII image. (The character "Δ" represents a blank and (CR) is a carriage-return.)

ΔP1,ΔPPΔ.;P3; 12 (CR)*

When the intrinsic is executed, NUM will contain the value 5, and the double-array PARMS will contain the following:

| | | |
|-------|---|---|
| %1201 | | |
| 2 | 6 | 0 |
| %1205 | | |
| 2 | 4 | 2 |
| %1211 | | |
| 0 | 0 | 1 |
| %1212 | | |
| 3 | 7 | 1 |
| %1216 | | |
| 2 | 2 | 3 |

(Byte address.)

(Length = 2; Alphanumeric characters; 0th delimiter in DELS.)

(Omitted Parameter.)

Bits 0 8 11 15

EXECUTING MPE/3000 COMMANDS PROGRAMMATICALLY

The COMMAND intrinsic call allows the user to programmatically request the execution of an MPE/3000 command. The command image is passed to the intrinsic, which searches the system command dictionary for a command of the same name, and executes it. When command execution is completed, or when an error is detected during this execution, control returns to the calling process. Commands that cannot be executed programmatically are indicated in Appendix B.

NOTE: Warning messages issued by the command executor are transmitted to \$STDLIST, with no programmatic indication to the calling process.

PROCEDURE

`COMMAND (comimage,error,parm);`

BYTE ARRAY *comimage*;

INTEGER *error*;

INTEGER *parm*;

OPTION EXTERNAL;

The parameters for the `COMMAND` intrinsic call are

- comimage* A byte array that contains an ASCII string making up a command and parameters terminated by a carriage return. No prompt character, however, should be included in this string. The *comimage* array may be altered by the `COMMAND` intrinsic (for example, in upshifting), but will be returned in a form that can be re-submitted to this intrinsic without adjustment.
- error* A word to which any error code set by the command is returned. (This is the same error code that would appear on the job/session list device if the command was part of the job/session input stream.)
- parm* A location to which the number (index) of the erroneous parameter is returned. If no *parameters* are in error, *parm* contains zero. (This is the same parameter that would be noted on the job/session list device if the command was part of the job/session input stream.)

The following condition codes can result from the `COMMAND` intrinsic:

- CCE The command was successfully executed.
- CCL The command was an undefined command.
- CCG An executor-dependent error, such as an erroneous parameter, prevented execution of the command. The location specified by the *error* parameter contains the numeric error code. The location specified by *parm* (if not zero) contains the erroneous parameter element. The *error* and *parm* parameters are returned only if the condition code is CCG.

EXAMPLE:

Suppose the user wants to programmatically execute the command `:SHOWTIME`. All characters for this command except the prompting colon are contained in the byte-array `COMD`. Any error code is to be returned to `ECODE`. Since the `:SHOWTIME` command has no parameters, no information will be returned to `EPARM`. The user issues the following call:

COMMAND (COMD,ECODE,EPARM);

The date and time are printed on the job/session list device.

ENABLING AND DISABLING TRAPS

Whenever a major error occurs during the execution of a hardware instruction, a procedure from the System Library, or an intrinsic called by the user, normally the user's program is aborted and an error message is output. The user can, however, avoid immediate abort by arming any of three software traps provided by MPE/3000:

- The Arithmetic Trap, for hardware instruction errors
- The Library Trap, for errors detected during execution of a system library procedure
- The System Trap, for errors detected during execution of a callable system intrinsic

When an error occurs, the corresponding trap, if armed, suppresses output of the normal error message, transfers control to a *trap procedure* defined by the user, and passes one or more parameters describing the error to this procedure. This procedure may attempt to analyze or recover from the error, or may execute some other programming path desired by the user. Upon exiting from the trap procedure, control returns to the instruction following the one that activated the trap. (In the case of the library trap, however, the user can specify that his process be aborted when control exits from the trap procedure.) Trap intrinsics can be invoked from within trap procedures.

Note: The validity of a trap procedure, specified by the external-type label of the user's trap procedure (plabel), depends on the code domain of the caller's code and executing mode (privileged or non-privileged), and on the code domain of the plabel and the mode (privileged or non-privileged). The code domains are:

*PROG (User Program)
GSL (Group SL)
PSL (Public SL)
SSL (System SL, non-MPE segments)
MPSSL (System SL, MPE segments)*

If, at the time of arming a trap procedure, the code of the caller is

1. *Non-privileged in PROG, GSL, or PSL: plabel must be non-privileged in PROG, GSL, or PSL.*

97+62

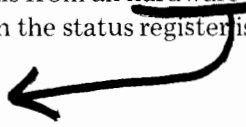
2. *Privileged in PROG, GSL, or PSL: plabel may be privileged or non-privileged in PROG, GSL, or PSL.*
3. *Privileged or non-privileged in SSL: plabel may be in any non-MPESSL segment.*

Arithmetic Trap

There are two levels of arithmetic traps: the hardware arithmetic trap set and the software arithmetic trap. Each trap in the hardware trap set detects a particular type of hardware error, such as division by zero or result overflow. The software trap, if armed, receives an internal interrupt signal from a hardware trap when an error is encountered, and transfers control to the user's trap procedure.

When the user's process begins execution, all hardware trap set interrupt signals are automatically enabled, but the software trap is disarmed, permitting any hardware error to abort the process. Through intrinsic calls, however, the user can alter the ability of the hardware trap set to send signals, and that of the software trap to receive a signal from any particular hardware trap. Only signals received and accepted by the software trap can invoke the user's trap procedure.

To enable or disable the internal interrupt signals from all hardware arithmetic traps, the user enters the ARITRAP intrinsic call. (The overflow bit in the status register is cleared, for the calling process.)

PROCEDURE *ARITRAP (state);* 

VALUE state;

LOGICAL state;

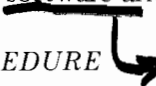
OPTION EXTERNAL;

In this intrinsic call, *state* is TRUE (Bit 15 = 1) to enable the signals from all traps, and FALSE (Bit 15 = 0) to disable them.

The following error conditions can result from the ARITRAP intrinsic:

- CCE Request is granted; the trap set signal was originally disabled.
- CCG Request is granted; the trap set signal was originally enabled.
- CCL (Not returned by this intrinsic.)

To arm the software arithmetic trap, the user enters the XARITRAP intrinsic call:

PROCEDURE *XARITRAP (mask,plabel,oldmask,oldplabel) ;* 

VALUE mask,plabel;

INTEGER mask,plabel,oldmask,oldplabel;

OPTION EXTERNAL;

The parameters of the XARITRAP call are

mask A word-mask that selects which hardware traps will invoke the software trap, and which will not. Only the 14 rightmost bits of the word forming the mask are used. (The setting of the other bits is not significant, but it is recommended that they be set to zero. Thus, octal values up to %37777 are suggested for this parameter.) If a bit is on, the corresponding hardware trap activates the software trap; otherwise, it does not. If all bits are set to zero, the software trap is *disarmed*.

| Bit | Hardware Error Trap |
|-----|-----------------------------------|
| 15 | *Floating-point divide by zero |
| 14 | Integer divide by zero |
| 13 | *Floating-point underflow |
| 12 | *Floating-point overflow |
| 11 | Integer overflow |
| 10 | Extended-precision overflow |
| 9 | Extended-precision underflow |
| 8 | Extended-precision divide by zero |
| 7 | * Decimal overflow |
| 6 | Invalid ASCII digit (CVAD) |
| 5 | Invalid decimal digit |
| 4 | Invalid source word count (CVBD) |
| 3 | Invalid decimal operand length |
| 2 | Decimal divide by zero |

plabel The external-type label of the user's trap procedure. If the value of this entry is 0, the software trap is *disarmed*. (The external-type label of the procedure, which resides in the segment transfer table of the procedure's code segment, is passed as a parameter (in SPL/3000) by placing a @ before the procedure name, as shown in the example below.)

oldmask A word in which the value of the previous *mask* is returned to the user's program.

oldlabel A word in which the previous *plabel* is returned to the user's program. If no *plabel* existed previously, 0 is returned.

The following condition codes can result from the XARITRAP intrinsic call:

CCE Request granted, software trap armed.

CCG Request granted; software trap disarmed.

CCL Illegal *plabel* (refer to NOTE under Enabling and Disabling Traps, page 8-26); no action taken.

EXAMPLE:

To arm the software arithmetic trap so that only floating-point division by zero alone causes a branch to the post-trap procedure `RETRY`, the user can enter the following call. The word-mask previously specified is returned to the location `OMASK`, and the previous label is returned to the location `OLABEL`.

`XARITRAP (Z1,0RETRY,OMASK,OLABEL);`

Library Trap

The software library trap reacts to major errors that occur during execution of procedures from the System Library. When the user's program begins execution, this trap is automatically disarmed. The user can arm (or disarm) it with the `XLIBTRAP` intrinsic call. When armed, the library trap passes control to a trap procedure in the event of an error. This procedure, in turn, returns to the user's program four words containing the stack marker created when the library procedure was called by the user's program. In addition, the trap procedure returns an integer representing the error number. When the procedure is completed, it either transfers control to the instruction following that which caused the error or aborts the job/session at the user's option. The trap procedure is defined by the user, but it must conform to the special format discussed in the manual *HP 3000 Compiler Library*.

The format of the `XLIBTRAP` intrinsic call is

PROCEDURE

`XLIBTRAP (plabel,oldplabel) ;`

VALUE *plabel*;

INTEGER *plabel,oldplabel*;

OPTION EXTERNAL;

plabel The external-type label of the user's trap procedure. If the value of this entry is 0, the trap is *disarmed*. *See LABEL on page 8-32.*

oldlabel A word in which the previous *plabel* is returned to the user's program. If no *plabel* existed previously, 0 is returned.

These condition codes can result from the `XLIBTRAP` intrinsic call:

`CCE` Request granted, trap armed.

`CCG` Request granted, trap disarmed.

`CCL` Illegal *plabel*, no action taken.

EXAMPLE:

To arm a disarmed library trap so that, if a library procedure error occurs, control is transferred to the post-trap procedure *RTNX*, the user enters the following call. (The old label of the trap procedure is returned to *OLDLAB*.)

`XLIBTRAP (RTNX,OLDLAB) ;`

System Trap

The software system trap reacts to errors occurring in intrinsics called by the user's program. Typical errors are

- Illegal access (an attempt by the user to access an intrinsic for which he does not have access capability).
- Illegal parameters (the passing to an intrinsic of parameters that are not defined for the user's environment).
- Illegal environment (the DB-register is not currently pointing to the user's stack area).
- Resource violation (the resource requested by the user is either illegal or outside the constraints imposed by MPE/3000).

When the user's program begins execution, the system trap is automatically disarmed. When armed by the *XSYSTRAP* intrinsic call and subsequently activated by an error, the trap transfers control to a trap procedure.

The system trap is armed or disarmed by the *XSYSTRAP* intrinsic call:

PROCEDURE *XSYSTRAP (plabel,oldlabel) ;*

VALUE plabel;

INTEGER plabel,oldlabel;

OPTION EXTERNAL;

plabel The external-type label of the user's trap procedure. If the value of this entry is 0, the software trap is disarmed.

oldlabel A word in which the previous *plabel* is returned to the user's process. If no *plabel* existed previously, 0 is returned.

These condition codes can result from the XSYSTRAP intrinsic call:

- CCE Request granted, trap armed.
- CCG Request granted, trap disarmed.
- CCL Illegal label, no action taken.

CONTROL-Y Traps

In addition to the error traps just discussed, a user running an interactive process can arm a special trap that transfers control from the currently-executing program or procedure to a trap procedure whenever a CONTROL-Y subsystem break signal is entered from the terminal during a session. (On most terminals, the signal is transmitted by striking the Y-key while depressing the control key.) When more than one process is currently running within the user's process tree structure, the CONTROL-Y signal interrupts the last process to arm the trap. (The trap is armed by issuing the XCONTRAP intrinsic call.)

When a process is interrupted by a CONTROL-Y signal, the following occurs:

1. The input/output transactions pending between the process and the terminal are halted and flagged as though all were completed successfully.
2. Control is transferred to the trap procedure defined by the user, with which he can now interact. (The trap procedure executes in the same mode (privileged or non-privileged) as the user program that was interrupted.)
3. Control returns from the trap procedure to the interrupted program or procedure. If the interrupted program or procedure was awaiting completion of input/output (reading from or writing to the terminal) when the CONTROL-Y signal was received, the FREAD or FWRITE intrinsic that was being executed is flagged as successfully completed when control returns from the trap procedure. If the CONTROL-Y signal was received during reading, the number of characters typed in *before* this signal is returned to the user as the value of FREAD. (The carriage position is unchanged.)

(I/O)
NOTE!

If the user sends another CONTROL-Y signal, it will be ignored unless a call to the RESETCONTROL intrinsic was issued at some point prior to the signal.

If the user sends a CONTROL-Y signal while MPE/3000 system code is executing on his behalf, no interrupt occurs until the process resumes execution of the user's code; then control transfers to the trap procedure. This protects the MPE/3000 System operation.

When a session is initiated, the CONTROL-Y trap is disarmed. The user arms this trap by issuing the XCONTRAP intrinsic call. This call takes effect on the file \$STDIN, when this filename indicates a terminal.

PROCEDURE

XCONTRAP (plabel, oldplabel);

VALUE plabel;

INTEGER plabel, oldplabel;

OPTION EXTERNAL;

The parameters are

- plabel* The external-type label of the user's trap procedure. If the value of this entry is 0, the software trap is disarmed.
- oldplabel* A word in which the previous *plabel* is returned to the user's process. If no *plabel* existed previously, 0 is returned.

These condition codes can result from the XCONTRAP intrinsic:

- CCE Request granted, trap armed.
- CCG Request granted, trap disarmed.
- CCL Illegal *plabel* (refer to NOTE under *Enabling and Disabling Traps*, page 8-26), or \$STDIN is not a terminal; no action taken.

To reset the terminal so that another CONTROL-Y signal can be accepted, the user calls the RESETCONTROL intrinsic. To take effect, this intrinsic must be called after the trap procedure is entered.

PROCEDURE

RESETCONTROL;

OPTION EXTERNAL;

The following condition codes are returned by this intrinsic:

- CCE Request granted.
- CCG (Not returned by this intrinsic.)
- CCL The request was not granted because the trap procedure was not invoked.

Trap Procedure Execution

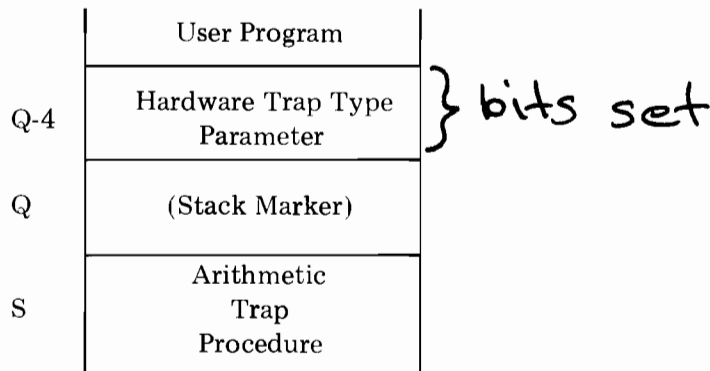
When writing trap procedures, the user should bear the following information in mind.

ARITHMETIC TRAPS. When a software arithmetic trap procedure is executed, the Index register contains the word of code being executed when the trap occurred. This information (plus, if necessary, the right stackop bit in the stacked status word) can be used to identify the offending instruction. A one-word parameter is available in (Q-4) in which certain bits indicate the type of hardware trap invoked. The various traps leave the parameter in Q-4 as follows.

Standard Traps

- Bit 15 = Floating Point Divide by 0
- 14 = Integer Divide by 0
- 13 = Floating Point Underflow
- 12 = Floating Point Overflow
- 11 = Integer Overflow

A return from the trap procedure (through an (EXIT 1) instruction) will resume execution in the user code domain at the instruction following that which activated the trap procedure. The condition of the stack when the trap procedure is invoked is



Extended Precision Floating Point Traps

- Bit 10 = Extended Precision Overflow
- 9 = Extended Precision Underflow
- 8 = Extended Precision Divide by 0

The address of the result operand is left on the stack in Q-5. An (EXIT 2) return will resume execution in the user code domain at the instruction following the one which caused the trap. The condition of the stack when the trap procedure is invoked is

| | |
|-----|---------------------------------|
| | User Program |
| Q-5 | Result Address |
| Q-4 | Hardware Trap Type Parameter |
| Q | (Stack Marker) |
| S | Arithmetic Trap Procedure |

Commercial Instruction Traps

- Bit 7 = Decimal Overflow
- 6 = Invalid ASCII Digit (CVAD)
- 5 = Invalid Decimal Digit
- 4 = Invalid Source Word Count (CVBD)
- 3 = Invalid Decimal Operand Length
- 2 = Decimal Divide by 0

The parameters stacked for the execution of the instruction are left on the stack below Q-4 and the SDEC specified in the opcode is ignored. To return properly the trap handler must examine the opcode (found in the Index Register) to determine the proper stack decrement to use on exit. The condition of the stack when the trap procedure is invoked is

| | |
|-----|---------------------------------|
| | User Program |
| | Stacked Operands |
| Q-4 | Hardware Trap Type Parameter |
| Q | (Stack Marker) |
| S | Arithmetic Trap Procedure |

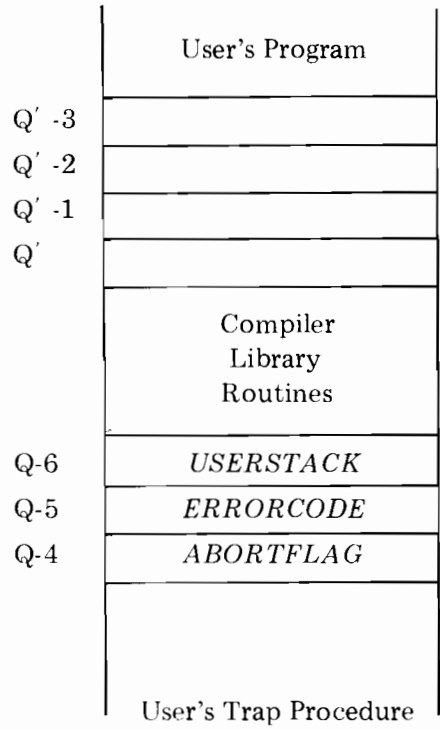
A suitable declaration for a user's arithmetic trap procedure might be

```

PROCEDURE ARITHMETICTRAP (parameter);
VALUE parameter;
LOGICAL parameter;
    
```

} EXAMPLE
~

LIBRARY TRAPS. When a library trap procedure is invoked, the condition of the stack is



- USERSTACK* A word pointer to the base of the stack marker placed on the stack when the user's program called the compiler library.

- ERRORCODE* A number indicating the type of compiler library error, described in *HP 3000 Compiler Library*.

- ABORTFLAG* A value set before the user exists from the trap procedure. If TRUE, the compiler library aborts the program with the standard error message (just as if no trap procedure had been executed). If FALSE, the compiler library does not abort the program and no error message is printed; in this case, the compiler library attempts error-recovery.

A suitable declaration for a user's library trap procedure is

```

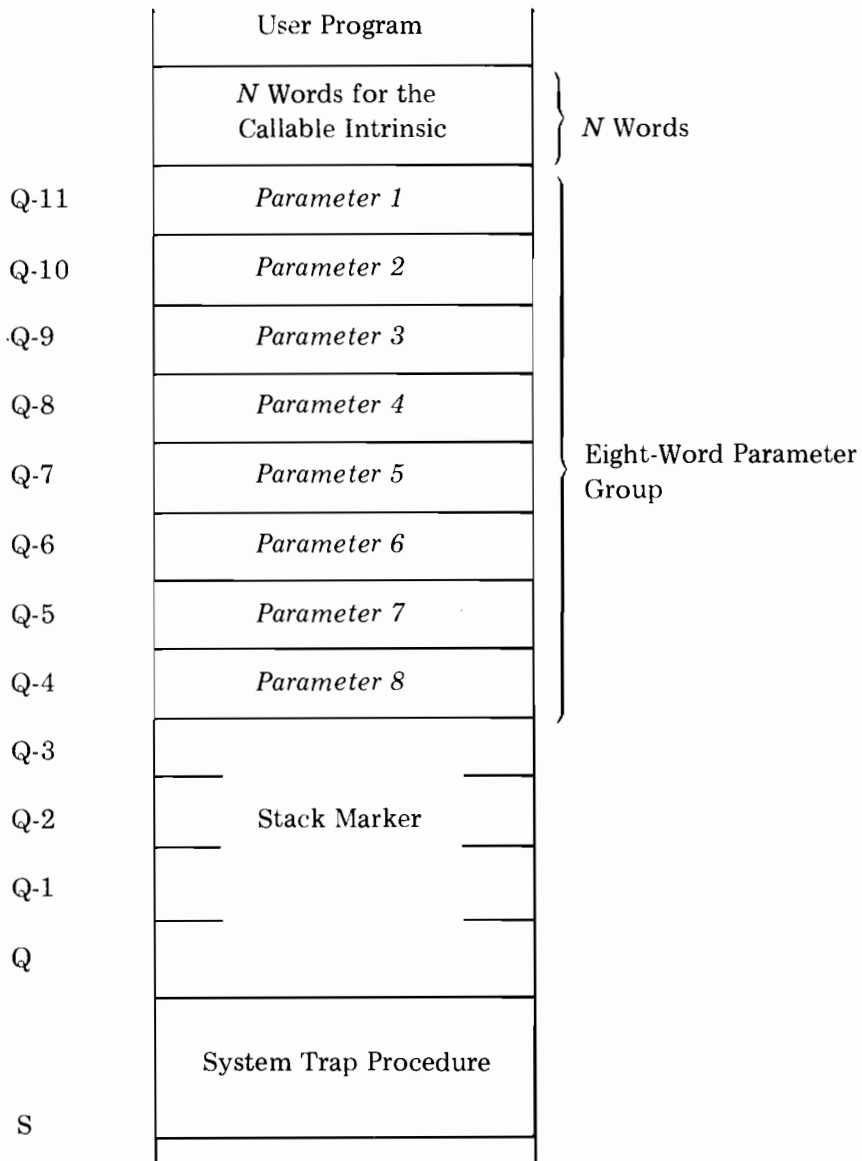
PROCEDURE LIBRARYTRAP (USERSTACK,ERRORCODE,ABORTFLAG);
ARRAY USERSTACK;
INTEGER ERRORCODE;
LOGICAL ABORTFLAG;

```

When execution of the library trap procedure is complete, control returns to the library routine that was interrupted.

System Traps

When a system trap procedure is executed because of an abort condition arising in a callable system intrinsic, the stack is readjusted to provide an eight-word parameter group between the intrinsic parameters and the stack marker.



The format of the eight-word parameter group in Q-4 through Q-11 is

| Bits | 0 | 9 | 10 | 15 | |
|------|-----|---|-----|----|--------------------|
| Q-11 | I | | N | | <i>Parameter 1</i> |
| Q-10 | P | | | | <i>Parameter 2</i> |
| Q-9 | P-1 | | E-1 | | <i>Parameter 3</i> |
| Q-8 | P-2 | | E-2 | | <i>Parameter 4</i> |
| Q-7 | P-3 | | E-3 | | <i>Parameter 5</i> |
| Q-6 | P-4 | | E-4 | | <i>Parameter 6</i> |
| Q-5 | P-5 | | E-5 | | <i>Parameter 7</i> |
| Q-4 | P-6 | | E-6 | | <i>Parameter 8</i> |
| | | 7 | 8 | | |

- I Intrinsic number, as defined in Figure 10-4, Section X.
- N Number of callable intrinsic parameters. (To resume execution in the user code domain, an EXIT N+8 instruction should be executed.)
- P Additional parameter information.
- P-1 through P-6 Parameters modifying the error bytes, described below. If no modifying parameter is present, the corresponding parameter byte is set to zero.
- E-1 through E-6 Error bytes, containing the error codes noted in Section X. The last error code present is delimited by the value of zero in the following error byte.

With these parameters, the trap procedure may take any recovery action necessary--write messages, produce selective dumps, set error-indication flags, or allow interactive debugging. Finally, the procedure may either call the TERMINATE intrinsic or issue an (EXIT N+8) instruction to return to the user's program (at the location following that where the trap was invoked), with appropriate error indications.

A sample declaration for a system trap procedure, and an example of how one might issue an EXIT N+8 instruction follow:

```
PROCEDURE      SYSTEMTRAP      (PARAMETER1,PARAMETER2,PARAMETER3,
                                PARAMETER4,PARAMETER5,PARAMETER6,
                                PARAMETER7,PARAMETER8);

VALUE          PARAMETER1,PARAMETER2,PARAMETER3,PARAMETER4,
                PARAMETER5,PARAMETER6,PARAMETER7,PARAMETER8;

LOGICAL        PARAMETER1,PARAMETER2,PARAMETER3,PARAMETER4,
                PARAMETER5,PARAMETER6,PARAMETER7,PARAMETER8;

BEGIN

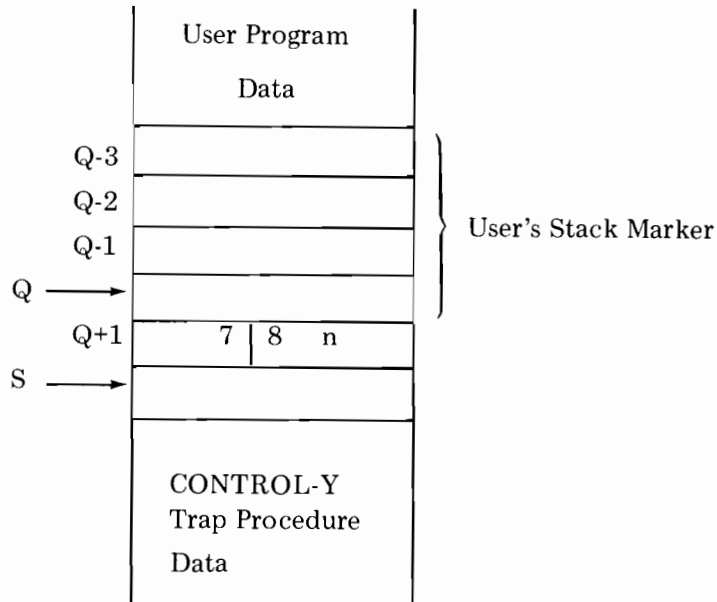
    INTEGER N;
        .
        .
        .
    << USER MAY OUTPUT MESSAGES >>
        .
        .
        .
    N:=PARAMETER1  LAND%37; <<N=NUMBER OF PARAMETERS
                                PASSED TO CALLABLE
                                INTRINSIC >>

    TOS:=N+%31410;          <<PUT "EXIT N+8" ON TOP
                                OF STACK >>

    ASSEMBLE (XEQ 0);      << EXECUTE "EXIT N+8" ON
                                TOP OF STACK >>

END;
```

CONTROL-Y Traps. A CONTROL-Y trap procedure assumes control when a CONTROL-Y entry interrupts the user's program. An (EXIT N) instruction later returns control to the instruction in the user's code domain following the last instruction executed before the CONTROL-Y trap procedure was invoked. When the trap procedure is invoked, the condition of the stack is as follows:



When the first instruction in the trap procedure is executed, the Q-Register points to the user's stack marker and the S-Register points to Q+2. The trap procedure should not write data in the rightmost byte of the word Q+1, because it is used to exit back to the interrupted code. (The value n is the N parameter in the (EXIT N) instruction, which must be placed on the stack as follows.)

TOS :=%31400 + N

The instruction is then executed (by an XEQ instruction).

NOTE: Users with the Privileged-Mode Optional Capability should be aware of the following:

1. *If the user's interrupted code was executing in privileged mode, his trap procedure must also be executed in privileged mode (and must thus have the privileged mode capability).*
2. *When a user's process is executing in privileged mode, and a CONTROL-Y signal invokes a trap procedure, the trap procedure is entered with the same DB register setting in effect when the signal was received. Thus, if the DB register is pointing to an extra data segment rather than the user's stack when a CONTROL-Y signal is received, it will continue to point to that extra data segment when the trap procedure is entered.*

A suitable declaration for a CONTROL-Y trap procedure is

PROCEDURE CONTROLYTRAP;

CHANGING STACK SIZES

When a user prepares or executes a process, he specifies (or allows MPE/3000 to assign by default) the size of the stack (Z-DB) area and the user-managed (DL-DB) area within the stack segment. Once the process begins execution, the user can programmatically change the size of these areas to meet new requirements as they arise, by altering the register off-sets Z-DB or DL-DB. For example, the user typically expands the size of these areas when he finds, during process execution, that the sizes initially specified were not sufficient for his data requirements. Conversely, he might contract the size of either of these areas should his process no longer require large amounts of space for data. These changes are requested through the intrinsics described below.

If the user intends to dynamically expand or contract these areas, he *must* specify, at the time the stack is created, the anticipated maximum size of the stack segment; this value is used by MPE/3000 in allocating disc storage. The maximum stack size value can be specified in the *segsz* parameter of the :PREP, :PREPRUN, and :RUN commands, or in the *maxdata* parameter of the CREATE intrinsic (used by programmers with the *Process-Handling Optional Capability* to create processes).

NOTE: When the stack segment belonging to a process running in privileged mode is frozen in main memory, either implicitly (when a user's process interfaces directly with the input/output system), or explicitly (by a direct call to appropriate uncallable system intrinsics), the intrinsics to change the register offsets DL-DB or Z-DB cannot be executed. When these intrinsics are called under such circumstances, a special FROZEN STACK error code is returned to the calling process, which may then attempt recovery. In general, this error code implies that the user should wait until the stack is unfrozen before re-issuing the intrinsic call.

Changing the DL-DB Area Size

The user can expand or contract the size of the current DL-DB area by altering the register off-set of the DL-address from the DB address (DL-DB). He does this by calling the DLSIZE intrinsic. This moves the current DB address and all following data forward (during expansion) or backward (during contraction) within the stack segment. All current information in the DL-DB area is saved. If the DL-DB size requested exceeds the maximum size permitted by the Z-DL (stack) area, only the maximum size permitted is granted. The intrinsic format is

INTEGER PROCEDURE

DLSIZE (size);

VALUE size;

INTEGER size;

OPTION EXTERNAL;

The size actually granted is returned to the calling process as the value of DLSIZE.

The *size* parameter specifies the new value of DL-DB. It is an integer less than or equal to zero. All increments or decrements to the DL-DB area are made in groups of 128 words. Thus, the size actually granted will depend on the currently existing size of the DL-DB area. If *size* is zero, DL is re-set to the value originally assigned when the process was created.

The following condition codes may result:

- CCE Request granted.
- CCG The requested size exceeded the maximum limits allowed for the Z-DL area (Z-DL offset). This maximum limit is granted, and its value is returned to the calling process.
- CCL An illegal *size* parameter was specified; the original size assigned when the stack segment was created is granted. The CCL condition code also results when a FROZEN STACK exists, but, in addition, DLSIZE is replaced by a positive value as a special error indicator.

EXAMPLE:

The user issues this intrinsic call to expand his DL-DB area to 300 words.

DLSIZE (-300);

Because DL-DB space is allocated in increments of 128 words, the amount of space actually granted may exceed 300 words, depending on the DL-DB setting prior to the call.

Changing the Z-DB Area Size

The user alters the size of the current Z-DB area by altering the register off-set of the Z address from the DB-address (Z-DB). He does this by calling the ZSIZE intrinsic. This moves the Z address forward (expansion) or backward (contraction). If the Z-DB area size requested exceeds the maximum size permitted for the Z-DL (stack area), only the maximum size allowed is granted. The format of the intrinsic is

INTEGER PROCEDURE *ZSIZE (size);*

VALUE size;

INTEGER size;

OPTION EXTERNAL;

The size actually granted is returned to the calling process as the value of ZSIZE.

The size parameter specifies the new register offset (in words) for Z-DB. This must be an integer value greater than or equal to zero. All changes to the Z-DB area are made in increments or decrements of 128 words, and hence the size actually granted may differ from the size requested.

The following condition codes may be returned:

- CCE Request granted.
- CCG The requested size exceeded the maximum limits of the Z-DL (stack) area. This maximum limit is granted, and its value is returned to the calling process.
- CCL An illegal *size* parameter, less than (S-DB)+64 words, was specified. This minimum value is assigned by default. The CCL condition code also results when a FROZEN STACK exists, but, in addition, ZSIZE is replaced by a negative value as a special error indicator.

REQUESTING A PROCESS BREAK

During a session, the user can interrupt an executing process, transferring control to the MPE/3000 Command Interpreter. This operation is called a *process break*. It allows the user to enter certain MPE/3000 commands, perhaps to create a file, transmit an informal message, or terminate a process. (The commands permitted during a break are designated in Appendix B.) The user initiates a break by pressing the *break* key on the terminal.

During a session, the user can also initiate a break programmatically, by entering the following intrinsic call (which requires no parameters). (This intrinsic is not valid in a job.)

```
PROCEDURE CAUSEBREAK;  
OPTION EXTERNAL;
```

A secondary entry-point to the CAUSEBREAK intrinsic, CAUSEBREAK', coincides with the main entry-point.

This intrinsic returns the following condition codes:

- CCE Request granted.
- CCG (Not returned by this intrinsic.)
- CCL Request not granted because the intrinsic was not called from a *session*.

The user can resume execution of a process at the point where it was interrupted, with this command:

```
:RESUME
```

TERMINATING A PROCESS

The user can programmatically terminate or abort a process as follows.

Termination

Process termination (the type occurring after successful execution of a process) is requested with the following intrinsic call:

```
PROCEDURE TERMINATE;  
  
OPTION EXTERNAL;
```

The process and all of its descendents (and any extra data segments belonging to them) are deleted. (Programmers using the CREATE intrinsic call to initiate the process can specify that the father of the terminating process be automatically activated.) All files still open by the process are closed and assigned the same disposition they possessed when opened. A secondary entry-point to the TERMINATE intrinsic, TERMINATE', coincides with the main entry-point.

Abort

From within any process in a user process structure, the user can call an intrinsic that aborts this process. This intrinsic is the QUIT intrinsic, which also transmits an abort message to the calling process output device. This intrinsic is identical to TERMINATE, except that it is used for abnormal termination and consequently sets the job/session in an error state. (In batch jobs not containing the :CONTINUE command, this results in job termination when the entire program finishes.) The format of the QUIT intrinsic is

```
PROCEDURE QUIT (num);  
  
VALUE num;  
  
INTEGER num;  
  
OPTIONAL EXTERNAL;
```

In this call, *num* is an arbitrary user-specified number. When the QUIT intrinsic is executed, *num* is also output as part of the resulting abort message. The format of the abort message is described in Section X.

4-15
5-21
5-30

The user may abort the entire user-process structure (program) by calling the QUITPROG intrinsic. This destroys all processes up to, but not including, the job/session main process. (The job/session main process is set in the error state; in batch jobs not containing the :CONTINUE command, this terminates the job.) An abort message, as described in Section X, is transmitted to the job/session listing device. The QUITPROG intrinsic format is:

```
PROCEDURE QUITPROG (num);  
  
VALUE num;  
  
INTEGER num;  
  
OPTION EXTERNAL;
```

In this intrinsic, *num* is an arbitrary user-specified number. When the QUITPROG intrinsic is executed, *num* is output as part of the abort message.

SETTING BREAKPOINTS AND DISPLAYING/MODIFYING MEMORY OR REGISTER DATA

The user can establish breakpoints (halts) in his program or display and modify data in memory or in various registers by invoking the DEBUG intrinsic. This intrinsic enables both non-privileged and privileged users to interactively check out their operating environment. Non-privileged users are bounded in scope to their private code segments and data base. Privileged users have all the DEBUG capabilities of non-privileged users plus a number of DEBUG extensions described separately under the heading "Privileged Mode DEBUG Extensions" in Section XIV of this manual. The intrinsic format is:

```
PROCEDURE DEBUG;  
  
OPTION EXTERNAL;
```

Invoking DEBUG

The DEBUG intrinsic can be invoked in three ways:

1. By direct external call to the intrinsic (read/write access to program file not necessary).
2. By reaching a previously established breakpoint within the user's program.
3. By an optional parameter in the :RUN or :PREPRUN commands or CREATE intrinsic, which sets a breakpoint on the first executable instruction of the program.

Upon entry, validity checks ensure that the user has an interactive device and (if not a direct external call) read/write access to the program file. If not, control is returned immediately to the user. Hence, a direct call from a user running in batch mode is essentially regarded as a null statement. (The DEBUG intrinsic does not change the condition code.)

Following these checks, a message is printed on the terminal; either

DEBUG segment.offset

if a direct call to DEBUG, or

BREAK segment.offset

if entered by a breakpoint.

(*Segment* specifies the logical program segment being executed; *offset* is the current relative offset of the call to DEBUG within that segment.) (Logical program segments are discussed in Sections IV and VII; the *segment* parameter is actually the logical segment number of the desired segment, as reflected in the listing requested through the *PMAP* parameter of the *:PREP* or *:PREPRUN* command.)

The DEBUG procedure prompts the user for a DEBUG command by printing a question mark (?) as a prompt character. Such commands, discussed below, can be used to establish breakpoints, request displays, and change stack or register contents.

Each command is verified for legal syntax and valid content, both statically and during execution. Upon detecting an error, the operation is terminated and an error message is printed. The message consists of a code word followed by a number (*nn*) indicating the character position in the command where the error was detected. The error message format is:

code nn

The following error codes are possible:

| Code | Meaning |
|--------|---------------------------------------------------------------------------|
| SYNTAX | Invalid syntax |
| NO-NO | Invalid information provided |
| BOUNDS | Bounds violation |
| FULL | Breakpoint table full |
| CHECK | New breakpoint location(s) conflicts with a previous breakpoint location. |

Upon entry, all messages (including prompt), commands, and listing output due to executing commands are directed to/from the interactive logical device. It is possible to re-direct only the listing output to another list device, such as line printer, through the L command (described later).

The instruction where a breakpoint occurred is not executed before entry to DEBUG.

DEBUG Command Format

A DEBUG command consists of:

- A *question mark* (used as a command identifier), printed automatically by DEBUG.
- A *command name* (mnemonic) consisting of a single letter (in most cases).
- A *parameter list* (in most cases).

Any number of optional spaces can be embedded *anywhere* in the command, even between digits in a parameter list, to provide a free and flexible format. Commands may be up to 68 characters long. The entire command is terminated by a carriage return.

All numbers entered or used in DEBUG parameters are by default in octal representation. However, decimal or ASCII numbers may be specified. A # character preceding a number specifies single-precision decimal (e.g., #987). A pair of quotes enclosing one or two ASCII characters specifies the numerical equivalent of those characters (e.g., "A" is %101; "AB" is %40502). A % character may optionally be used to designate octal.

Where parameters may be given as an expression (e.g., for *segment*, *offset*, etc.), the calculated result of each operation within the expression must be a single-precision number. The definition of an expression is as follows:

```
octal      := [%] octaldigit . . .
decimal    := #decimaldigit
ascii      := "[asciicharacter] [asciicharacter]"
number     := octal|decimal|ascii|null
factor     := number|(expression)
term       := factor [ {*/} factor ]
expression := term [ {+|-} term ]
```

Setting Breakpoints

The user establishes a breakpoint within his program by entering the B command:

?B [segment.] offset[, [segment.] offset] . . .

segment The logical code segment to contain the breakpoint. If omitted, the current segment applies.

offset The relative offset (displacement from the start of the segment) of the breakpoint.

This command sets a breakpoint at each location specified by a *[segment.] offset* parameter. A B@ command will display all breakpoints belonging to the process.

The B@ display format is:

LCST=xxx, P=xxx, [@], [conditional count/current count]

where LCST is the logical code segment number, P is the offset, @ indicates permanent breakpoint (see below), and count values are applicable to permanent/conditional breakpoints (see below).

When a breakpoint is reached during execution of the user's program, it is also cleared. However, a breakpoint can be made permanent (i.e., can be cleared only by an explicit C command) by appending ":@" to the parameter, as follows:

[segment.] offset:@

Further, a breakpoint can be made "conditionally" permanent (will break only after a specified number of executions) by appending ":@expression" to the parameter, as follows:

[segment.] offset:@expression

Or, the conditional feature may be used without permanence (will break only after a specified number of executions, then delete) by appending ":%expression" to the parameter, as follows:

[segment.] offset:%expression

It is also possible to set breakpoints in library segments, which are referenced directly or indirectly by the program (provided the segments are non-privileged) by preceding the parameter with G (group) or P (account) as follows:

G[segment.] offset
P[segment.] offset

NOTE: When using permanent/conditional breakpoints, be sure that the breakpoint location transfers control only to the next location in sequence; e.g., do not set this kind of breakpoint on a branch instruction. Since a permanent/conditional breakpoint uses two sequential instructions, it is not possible to set another breakpoint on either of these two locations. Also note that setting a breakpoint temporarily modifies the instruction at which the breakpoint occurs; thus, users are cautioned to ensure that this will not have any adverse side-effect upon their programs.

EXAMPLE:

The following B command establishes a breakpoint at Location 75 of Segment 3, Location 20 of the current program segment (permanent — break every time), Location 44 of program segment 4 (conditional — break and delete after 20th time), and Location 22 of group library segment 1 (permanent and conditional — break every 12th time).

```
?B 3.75, 20:@, 4.44:20, G1.22:@12
```

Clearing Breakpoints

A program breakpoint can be cleared by entering the C command:

```
?C [segment.] offset[, [segment.] offset] . . .
```

segment The logical program segment containing the breakpoint. If omitted, the current segment applies.

offset The relative offset of the breakpoint from the start of the segment.

To clear all breakpoints in a program, the format is:

```
?C@
```

EXAMPLE:

The following C command clears breakpoints at Location 50 in the current program segment, Location 33 in program segment 2, and Location 77 in public library segment 4.

```
?C 50, 2.33, P4.77
```

Resuming Program Execution

To resume execution of the program, the user enters either the **R** command (Resume) or the **E** command (Exit). The difference is that **R** provides an option to establish another breakpoint, whereas **E** provides options to delete a specified number of procedure parameters from the stack, or to terminate the program.

The **R** command is specified as follows:

?R [*segment.*]*offset*

segment The logical program segment to contain the new breakpoint. If omitted, the current segment applies.

offset The relative offset, from the beginning of the segment, of the new breakpoint.

If the optional parameter list is included in the **R** command, the new breakpoint is established before the user's program resumes execution. Rules for specifying the parameter (e.g., for conditional/permanent breakpoints) are the same as for the **B** command. If the parameter list is omitted, execution resumes without establishment of a new breakpoint. Control returns either to the instruction following a direct call to **DEBUG** or to the instruction which generated a break to **DEBUG**.

The **E** command is specified as follows:

?E [*expression*]

expression An expression which specifies the number of procedure parameters to be deleted from the stack. This expression cannot be greater than the value (S-Q).

The **E** command returns control to the appropriate instruction by performing an (EXIT N) instruction, where N is the result of the expression provided. An "E@" will terminate the program.

EXAMPLES:

To resume program execution and run until Location 100 of the current segment is encountered, the user enters:

```
?R 100
```

To delete 10 parameters from the stack and resume program execution, the user enters:

```
?E 10
```

Switching Display Output to a File

To switch all display output (e.g., as generated by T, DR, and D commands) to a file instead of to the interactive device, the user enters the L (List) command:

```
?L [filereference]
```

filereference The name (and optional group and account) of the file to which all display output is to be directed.

The L command opens the specified file, which then remains open until another L command is issued. If no *filereference* parameter is included, the file will be closed (assuming one is open) and the command again directs display output to the interactive device; an L0 has the same effect.

EXAMPLE:

To display output on the file named PRINTER, the user enters:

```
?L PRINTER
```

Displaying Register Contents

To display the current contents of the Q, S, X, ST (status), P, Z, and/or DL registers, the user enters the DR command in this format:

```
?DR [,register] [,register]. . .
```

register The register whose contents is to be displayed. This can be Q, S, X, ST, P, Z, or DL. If omitted, all registers are displayed.

If all registers are to be displayed, the logical code segment index (LCST) of the current program segment is also listed; the value is preceded by S if system SL, G if group SL, P if public SL, or blank if user program.

EXAMPLE:

To display the contents of the status and Z registers, the user enters:

```
?DR ST,Z
```

Displaying Memory Contents

To display memory contents of a specified number of locations relative to a given code base or data base, the D command is used in the following format:

?D [dispbse] [offset] [,count[,mode]]

dispbse One of the following stack relative display bases:

DB, DL, Q, S

or one of the following code relative display bases:

PB, P, PL

If no *dispbse* is specified, DB is assumed.

offset The offset relative to the display base which specifies the memory location at which the area to be displayed begins. It is written in this format:

[±] expression [:[±] expression] . . .

If no sign is given, positive offset is assumed. If the expression is followed by a colon, indirect addressing is indicated (relative to DB for stack relative display bases or PB for code relative display bases); may be followed by additional offset expression. Multiple levels of indirect are permitted.

count An expression (octal, decimal, or ASCII as noted above) which defines the number of memory locations to be displayed. If omitted, *count* is assigned a default value of 1.

mode A one-character specifier to indicate the representation mode for output values: O for octal, I for decimal, or A for ASCII. Default mode is octal.

The format of the display is up to eight location contents per line preceded by a representation of the address of the first location within the line.

EXAMPLES:

Several D commands and their resultant displays are shown below:

| <i>D Command</i> | <i>Item Displayed</i> |
|------------------|---------------------------------------------------------------------------------------------------------------------|
| ?D 100 | Location 100 (Relative to DB). |
| ?D Q-5 | Location Q-5. (If Q = 100, then this specifies Location 73.) |
| ?D Q-5: | Location pointed to by Location Q-5. (If Q = 100 and Location 73 contains 50, then this specifies Location 50.) |
| ?D Q-5:, 100,I | 100 (decimal) words pointed to by Location Q-5. |
| ?D PL-5:, 4 | First four instructions for STT entry 5 in the current code segment. (STT table begins, negative direction, at PL.) |

Modifying Register Contents

To modify the contents of the Q, S, X, ST, P, Z, and/or DL registers, the user enters the MR command in this format:

?MR [,register] [,register] . . .

(Omitting the parameter list indicates "all" registers.) New values are then requested, one at a time, through the interactive device in the form:

register = currentcontents :=

to which the user responds with an expression to specify the desired new contents, terminated by a carriage return. If the input is invalid, further := characters are output to receive the corrected value. A carriage return alone (i.e., null value input) preserves the current contents. A period (.) terminates any more requests for modification.

NOTE: The following restrictions apply to the MR command:

1. The register contents must be such that

$$DL \leq 0 \leq Q \leq S \leq Z$$

2. Only Bits 2 through 7 of the ST register can be changed.

EXAMPLE:

An MR command may result in the following interaction:

```
?MR
X    =444:=2           change to 2
ST   =140022:=       no change
DL   =177600:=500    illegal
      :=177300       good value
Q    =26:=           no change
S    =27:=26        new value of 26 and terminate modifications
```

Modifying Memory Contents

To modify the contents of a specified number of locations relative to a given base in the user's stack, the M command is used in the following format:

?M [modbase] [offset] [,count[,mode]]

modbase One of the following stack relative modification bases:

DB, DL, Q, S

If no *modbase* is specified, DB is assumed.

offset The offset relative to the modification base which specifies the memory location at which the area to be modified begins. It is written in this format:

[±]expression[:[[±]expression]] . . .

If no sign is given, positive offset is assumed. If expression is followed by a colon, indirect addressing is indicated (relative to DB); may be followed by additional offset expression. Multiple levels of indirect are permitted.

count An expression (octal, decimal, or ASCII as noted above) which defines the number of memory locations to be modified. If omitted, *count* is assigned a default value of 1.

mode A one-character specifier to indicate the representation mode for output values: O for octal, I for decimal, or A for ASCII. Default mode is octal.

New values are then requested, one at a time, through the interactive device in the form:

address currentcontent :=

to which the user responds with an octal expression to specify the desired new contents, terminated by a carriage return. If the input is invalid, further := characters are output to receive the corrected value. A carriage return alone (i.e., null value input) preserves the current contents. A period (.) terminates any more requests for modification. The *currentcontent* will be output in octal, decimal, or ASCII, depending on the *mode* specified.

EXAMPLE:

To modify four locations starting at DB+5 in the stack, the user enters:

?M 5,4

Requesting Trace of Stack Markers

The user can request a trace of the current stack marker contents by entering:

?T

This displays the stack marker Q location, the logical code segment number (LCST), and the P location for all stack markers.

EXAMPLE:

The T command below results in the following information:

?T

Q=40,LCST=G5,P=22 *Location for call to current procedure*

Q=12,LCST=0,P=5 *Location from which G5.22 was called*

Calculating an Expression

The user can request a display of the result of a given expression, in octal, decimal, or ASCII, in this format:

= expression[,mode]

mode The representation mode for the output value: O for Octal, I for decimal, A for ASCII; if omitted, octal result is given.

EXAMPLES:

*= 4+(5*2=3/(1+1))* *displays: =15*

= # 4+(55# 1?) - "A", I* *displays: =+479*

DUMPING THE STACK

The stack dump facility is composed of two complementary, independent features:

- A callable intrinsic that enables any program to selectively dump any part of the stack to any file.
- A mechanism that may be armed/disarmed by a command or intrinsic which will cause special actions upon a program abort: for batch jobs, parts or all of the stack are dumped on the standard list device; for an interactive session, an automatic call to DEBUG is generated.

Callable Stack Dump

The STACKDUMP intrinsic makes possible the dynamic monitoring of any or all variables of a program. As such, this facility may be regarded as a debugging aid. No special capability class is required to call STACKDUMP. The intrinsic format is:

| | |
|-----------------------------------|------------------------------------------------|
| <i>PROCEDURE</i> | STACKDUMP (filen,idnumber,flags,selec); |
| <i>BYTE ARRAY</i> | filen; |
| <i>LOGICAL</i> | flags; |
| <i>INTEGER</i> | idnumber(?) ; |
| <i>DOUBLE ARRAY</i> | selec; |
| <i>OPTION VARIABLE, EXTERNAL;</i> | |

This procedure contains a secondary entry point called STACKDUMP'.

filen A byte array that contains the file name of the file where the information is to be dumped.

If the secondary entry point is used to enter the procedure it is assumed that byte 0 of **filen** contains the file number of a file which has been successfully opened prior to the call to STACKDUMP. In this case, the file is not closed before going back to the user.

When **filen** contains the formal designator of the file, the file will be opened and closed by STACKDUMP.

idnumber An integer which is displayed in the header of the dump to help identify the printout.

flags A logical value which may be used to specify options as follows:

(14:1) = 1 Suppress the ASCII dump.

(15:1) = 1 Suppress the traceback of stack markers.

selec An array whose content specifies which stack areas are to be dumped. The format of the array is as shown in Figure 8-1. The array has no predetermined length; the first 0/-1 double word indicates the end of the array. Any entry for which the count is 0 is understood to be a "skip" (i.e., go to next double word element in list).

If **filen** is missing from the parameter list, the dump will take place on the standard list device. If **filen** is illegal (illegal file number or illegal file name) no dump takes place and CCL is returned.

In case a file number is passed (via STACKDUMP' entry point) the record length must be between 32 and 256 words and write access must be allowed to the file.

If STACKDUMP has to open and close the file, it is done with FOPTION=omitted and AOPTION=%2.

The default for **flags** is 0; that is, a corresponding ASCII dump will take place for all values dumped in octal, and a traceback of stack markers is displayed.

If the parameter **selec** is missing or if the first double word of the array is a stopper, then no dump (except for the header) occurs, except that if **flags(14:1)** is 0, the traceback of stack markers is displayed.

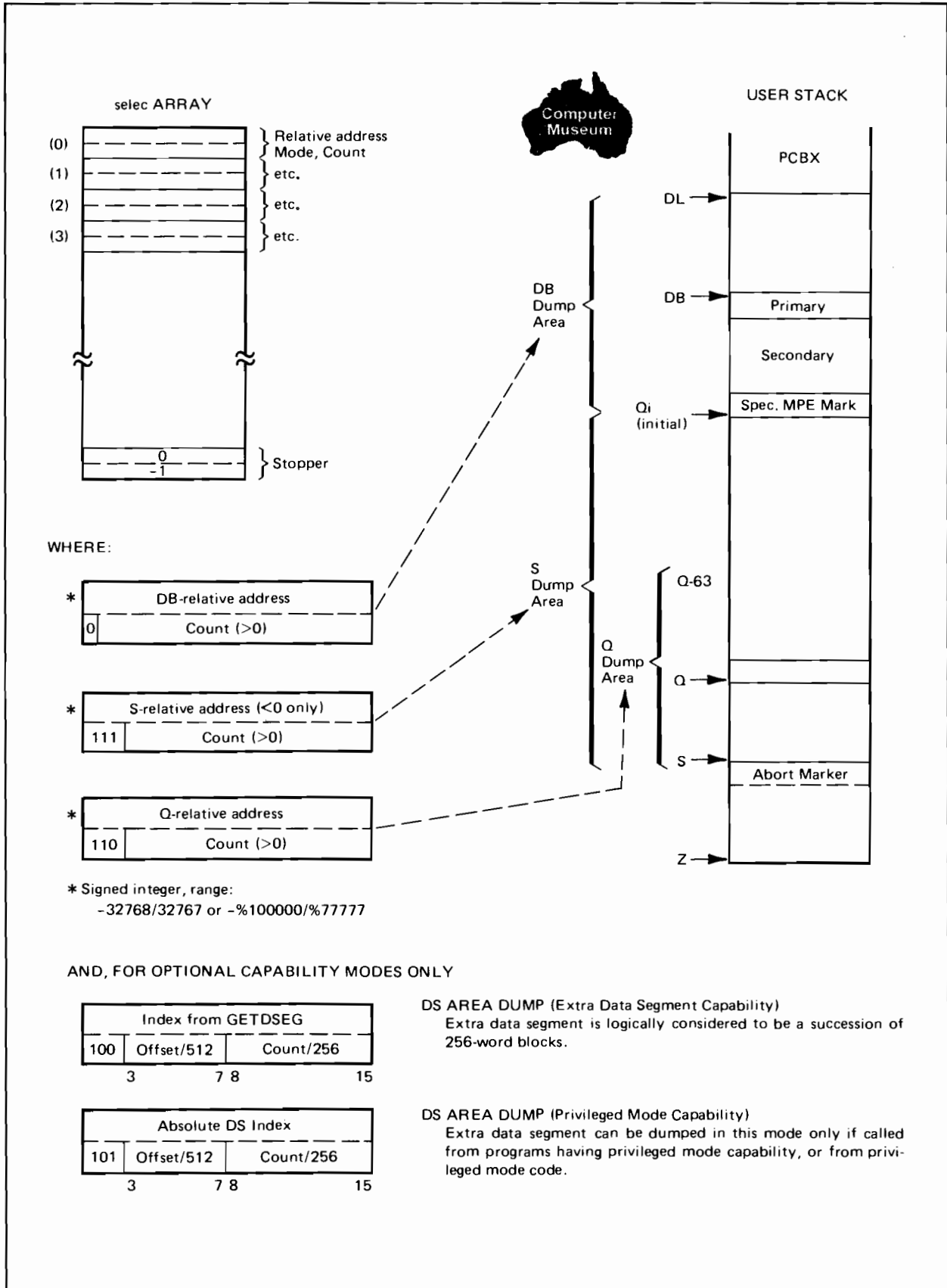


Figure 8-1. Stack Dump Modes and selec Array Format

If **idnumber** is missing from the parameter list, no identification number is displayed along with the header of the output.

When **STACKDUMP** is called, **DB** must be pointing at the stack.

The following condition codes apply:

- CCE Call is acceptable.

- CCG Bounds violation occurred; the dump was not completed.
 Record size was not between 32 and 256 words.

- CCL File system error (from opening, writing or closing of the file).
 The file error number is then returned in **idnumber**.

For each call to **STACKDUMP**, a header is output on the file:

```
***STACK DISPLAY***      IDNUMBER=
```

followed by the values of the registers at the time of the call, all being **DB** relative.

```
S=      DL=      Z=
Q=      P=      LCST=      X=
```

(All values are displayed in octal representation.)

This represents the extent of what is output in case the parameter is omitted and **flags (14:1) = 0**.

If **flags (14:1) = 1** then the following is output (traceback of the stack markers):

```
Q=      P=      LCST=(logical)      STAT=      X=
Q=      P=      LCST=(logical)      STAT=      X=
etc. . .
```

going from low address to high address in the stack. The traceback stops when $Q < 4$ or $Q > 2 * 15$ or if the **CST** is invalid for the process, or if $Q - (\Delta Q)$ falls beyond the initial **Q**.

The display continues with the output of the areas specified by the **selec** array. The areas are displayed in the order in which they are found in **selec**.

For each area/range specified, a bounds checking is performed to make sure that all values for the area are between **DL** and **S**. If this is not the case, then an appropriate message is output in lieu of the display of the area, and the dump proceeds for the next area.

Each line/record of output contains three fields:

- Address reference field
 - DB (DS) relative for DB (DS) type area
 - DB, and Q relative for Q type area
 - DB, Q and S relative for S type area
- Octal content of the stack (for easier reading, grouped in a multiple of 4 words of stack on each line/record).
- ASCII conversion of the above values, if so indicated by **flags**. A period (.) is output for nonprinting characters.

For example, if a Q type area is to be dumped on a terminal (72 bytes/record), starting at Q-3, with a count of (decimal) 10 words and no ASCII, the output is then:

| DB | Q | |
|-----|----|-------------------------------------------|
| 271 | -3 | xxxxx xxxxx xxxxx xxxxx xxxxx xxxxx xxxxx |
| 300 | 4 | xxxxx xxxxx xxxxx |

Abort Stack Analysis

This facility is different from the preceding in that it takes effect only in case of abnormal termination of a process. When the Abort Stack Analysis mechanism is armed for a process and if the process aborts, then a special action is taken by the operating system: If the process belongs to a batch job then the stack is dumped (in part or in whole) on the \$STDLIST device before the process terminates; if the process belongs to an interactive job (session), and before it takes any step towards its own termination, a call to DEBUG is performed thus letting the user analyze interactively the content of the stack and enabling him, at return time, either to continue in sequence (i.e., to abort the regular way) or to exit back to his main line of code in an attempt to recover.

The Abort Stack Analysis facility is incurred by any process:

- a. for which the facility has been armed (by SETDUMP command or intrinsic, to be defined later);
- b. which is aborted through one of the following ways:
 - Illegitimate stack overflow
 - MPE detected error in the call of an intrinsic (invalid parameter, invalid environment, lack of proper capability, etc.)

- Internal Interrupts
 - %11 Module error
 - %12 Parity error
 - %13 Miscellaneous error
 - %15 STT entry uncallable
 - %17 Traps
- Exit through an invalid marker (CST) which makes Absence Trap Routine fail;

- c. for which no user-provided procedure (XLIBTRAP or XARITRAP) is currently armed at the process level for that particular cause of abortion.

The mechanism is not effective under abnormal termination resulting from :ABORT, or =ABORTJOB or abortion of a process resulting from the prior abortion of its son or any member of its descendants.

Furthermore, in order to protect “proprietary” software no action (except for the dumping of the header and the stack marker trace) takes place for program files for which the user does not have read and write access.

If the abort occurs in the non MPE SL, no action (except for the dumping of the header and stack marker trace) will take place unless the user has privileged mode.

The mechanism is armed at process level, but the arming is propagated downwards from father to son to all processes subsequently created. The mechanism can also be disarmed.

Two commands are used to arm and disarm the facility. (The two corresponding intrinsics will be defined subsequently.) The arming command is:

```
:SETDUMP [stackrange[:ASCII]]
```

where *stackrange* can be one or more of the keywords *DB*, *ST*, or *QS*, submitted in any order, separated by commas, and defined as follows:

- DB* A DL to Qi dump (see Figure 8-1).
- ST* A Qi to S dump.
- QS* A Q-63 to S dump (ignored if *ST* is present in the parameter list).

Absence of all parameters results in the display of the set-up of the registers at the time of the abort, and stack marker trace.

The *ASCII* option, when required, causes an ASCII conversion of the octal content to be dumped along with the octal values.

The use of that command causes the Main Process to be armed as well as all processes created subsequently (through :RUN, :SPL, :SPLGO, etc.,) until such a time where it is reset through the following command:

:RESETDUMP

This command clears out the mechanism in the Main Process.

:SETDUMP and :RESETDUMP are allowed in Break Mode but do not modify the state of the processes already created.

No special capability is required for their use.

The mechanism is armed only between matching pairs of :SETDUMP and :RESETDUMP the job/session command stream.

Two :SETDUMP commands without an intervening :RESETDUMP is allowed and modifies the setting of the mechanism according to the latest command, except for the provision that if it is input in break mode it does not take effect until another process is initiated from the command interpreter.

A :RESETDUMP without a prior setting has no effect.

NOTE: In session mode, all parameters are ignored, and the only effect of the SETDUMP command is to arm the mechanism in order for the process to go to DEBUG in case of abortion.

The two intrinsics SETDUMP and RESETDUMP have the exact same effect on the caller process as the previous commands do on the Main Process. The format of the SETDUMP intrinsic is:

PROCEDURE

SETDUMP (flags);

VALUE flags;

LOGICAL flags;

OPTION EXTERNAL;

The procedure arms the Abort Stack Analysis facility of the caller process (and subsequent sons) according to the bit setting in *flags*.

| | |
|---------------------|----------------------------------------------------------|
| <i>flags (15:1)</i> | A DL to Qi dump (see Figure 8-1). |
| <i>(14:1)</i> | A Qi to S dump. |
| <i>(13:1)</i> | A Q-63 to S dump (ignored if <i>flags (14:1)</i> is on). |
| <i>(12:1)</i> | Ignored. |
| <i>(11:1)</i> | Suppress ASCII portion of dump. |

A 0 value for *flags* results in the display of registers and stack marker trace only.

In session mode the parameter list is ignored and the effect of the call is to get into DEBUG if and when the process is aborted.

The following condition codes apply:

- CCE Call is acceptable.
- CCG Mechanism already armed prior to call (now set up according to new specifications).
- CCL (Null.)

The format of the RESETDUMP intrinsic is:

```
PROCEDURE      RESETDUMP;  
OPTION EXTERNAL;
```

This procedure disarms the mechanism for the caller process but does not have any effect on the state of any of the processes of the family.

The following condition codes apply:

- CCE Call is acceptable.
- CCG Mechanism already disarmed prior to call (remains disarmed).
- CCL (Null.)

The facility can also be armed or disarmed by setting bits in the word *flags* of the CREATE intrinsic. In this case, it is armed for a process to be created, and it has no effect on the caller process. (See CREATE intrinsic definition.)

When the facility is entered in terminal session mode, the following title is printed:

```
***ABORT STACK ANALYSIS***
```

and DEBUG is then entered.

When the facility is entered from a batch job, the dumping of the stack takes place on \$STDLIST according to specifications passed through the SETDUMP command or intrinsic. Figure 8-2 illustrates a typical S-type dump (from Qi to S), with individual elements identified as follows:

- ① Abort message.
- ② Entry message for Abort Stack Analysis.
- ③ Register values at the time of abort.
- ④ Traceback of stack markers.
- ⑤ DB-relative address of first item on that line; succeeding items assume ascending addresses.
- ⑥ Octal dump of stack contents.
- ⑦ ASCII dump of stack contents; period represents non-printing character; blank represents ASCII blank.
- ⑧ Conclusion of job abort.

INTERPROCESS COMMUNICATION

The user can arrange for two processes belonging to the same job to communicate with each other through a *job control word*. This word is used primarily by systems programmers to enable a subsystem process to return information to the command executor that initiated that process. (Such a communication mechanism is used by the command executors for :RUN and various subsystem commands.) The general user may find this control word helpful in other applications.

Within the 16-bit control word, bit 0 is a sign bit used to notify the command executor (or other receiving process) of the normal termination (bit 0 = 0) or abort (bit 0 = 1) of the process. (This is the only HP convention governing the passing and interpretation of job control words between steps within a job.) Once bit 0 is set to 1, it can only be re-set by the actions of the :CONTINUE command. The remaining 15 bits can be used for whatever purpose desired, and can be set and re-set as necessary.

The following intrinsic call is used to set the bits in the job control word:

PROCEDURE *SETJCW (word);*

VALUE word;

LOGICAL word;

OPTION EXTERNAL;

In this call, *word* is a word whose bit contents, established by the user, are placed in the job control word. Bit 0 in *word* cannot be set to 0.

The SETJCW intrinsic does not change the condition code.

The next intrinsic call returns the complete job control word to the calling process:

LOGICAL PROCEDURE *GETJCW;*

OPTION EXTERNAL;

The job control word is returned as the value of GETJCW.

The GETJCW intrinsic does not change the condition code.

EXAMPLE:

Suppose that a user is writing a job where several processes pass information to each other through the job control word. In one process, the user transmits the contents of the word PROCLNK to the job control word. By HP convention, Bit 0 of this word signifies the successful execution or abort of the process setting the job control word, but the meaning of the remaining 15 bits must be established by the user's conventions. Process A sets the job control word to PROCLNK, as follows:

SETJCW (PROCLNK);

When Process B is executed, it obtains this current job control word through the GETJCW intrinsic. In this case, the contents of the job control word are returned to the word STORELNK.

STORELNK := GETJCW;

VERIFYING DIAGNOSTIC DEVICE ASSIGNMENT

Users running diagnostic programs must frequently acquire sole access to one or more devices normally managed by MPE/3000 in order to perform field maintenance tests on those devices while interfacing with them at the start input/output (SIO) level. The user can verify the assignment of such devices programmatically through the CHECKDEV intrinsic:

LOGICAL PROCEDURE *CHECKDEV (ldev,hdwradr);*

VALUE *ldev*;

INTEGER *ldev,hdwradr*;

OPTION EXTERNAL;

This intrinsic returns to the user's process a word containing one of these logical values (as the value of CHECKDEV).

TRUE, (Bit 15 = 1) if all units referenced are assigned to diagnostic programs.

FALSE, (Bit 15 = 0) if one or more units referenced are not yet so assigned.

The parameters are

ldev An integer indicating the logical unit number of the device to be isolated.

hdwradr An integer indicating the following:

0, to return the DRT and unit numbers of the device specified by *ldev*, and check the diagnostic assignment of this unit. When the intrinsic is executed, Bits 0-7 of the word *hdwradr* will contain the DRT number of the device reference, and Bits 8-15 will contain its unit number.

1, to return the DRT number only of the device, and check the diagnostic assignment of *all* devices assigned this DRT number. When the intrinsic is executed, Bits 0-7 of the word *hdwradr* will contain the DRT number, and Bits 8-15 will be set to zero.

The following condition codes apply:

CCE The requested logical unit is defined in MPE/3000.

CCG The logical unit number specified is invalid.

CCL (Not returned by this intrinsic.)

EXAMPLE:

To check the diagnostic assignment of a device whose logical unit number is 23, and to return its DRT and unit numbers to the word *INFOLNO*, the user issues this call. If the unit requested is assigned, the value of the word *ASGT* will be *TRUE*.

ASGT = CHECKDEV (23,INFOLNO)

INTRINSICS FOR COMPILER WRITERS

Programmers writing compilers may need to programmatically request certain operations on USL files. For these programmers, the following utilities are provided.

Initializing Buffers for USL Files

The user can initialize the first record (Record 0) of a USL file to the empty state by calling the *INITUSLF* intrinsic.

```
INTEGER PROCEDURE INITUSLF (uslfnm,rec0);  
VALUE uslfnm;  
INTEGER uslfnm;  
INTEGER ARRAY rec0;  
OPTION EXTERNAL;
```

This intrinsic returns (as the value of *INITUSLF*) an error number (if an error occurs); if no error occurs, no value is assigned to *INITUSLF*.

The parameters are

uslfnm A word identifier supplying the filename of the USL file.
rec0 A 128-word buffer to be initialized to the empty state, corresponding to the first record of the USL file (record 0).

The condition codes possible are

CCE Request granted.
CCG (Not returned by this intrinsic.)
CCL Request denied; an error number is returned as the value of *INITUSLF*. (The error numbers possible are listed at the end of this sub-section.)

Changing the Directory Block/Information Block Size on a USL File

The user can move the information block forward or backward on a USL file, thereby increasing or decreasing, respectively, the space available for the file directory block. (Notice that this does not change the overall length of the file.) This is done with the ADJUSTUSLF intrinsic.

```
INTEGER PROCEDURE      ADJUSTUSLF (uslfnm,records);  
  
VALUE uslfnm,records;  
  
INTEGER uslfnm,records;  
  
OPTION EXTERNAL;
```

This intrinsic returns (as the value of ADJUSTUSLF) an error number (if an error occurs); if no error occurs, no value is assigned to ADJUSTUSLF.

The parameters are

uslfnm A word identifier supplying the filename of the USL file.

records A signed record count. If *records* is greater than 0, the information block is moved forward in the USL file, increasing the space available for the directory block and decreasing that available for the information block. If *records* is less than 0, the information block is moved backward in the file, decreasing directory-block space and increasing information-block space.

The condition codes possible are

CCE Request granted.

CCG (Not returned by this intrinsic.)

CCL Request denied; an error number is returned as the value of ADJUSTUSLF. (The error numbers possible are listed at the end of this sub-section.)

Changing the Size of a USL File

The user can increase or decrease the length of a USL file by calling the EXPANDUSLF intrinsic:

```
INTEGER PROCEDURE      EXPANDUSLF (uslfnm,records);  
  
VALUE uslfnm,records;  
  
INTEGER uslfnm,records;  
  
OPTION EXTERNAL;
```

When this intrinsic is executed, a new USL file is created whose length is *records* longer (or shorter) than the USL specified by *uslfnm*. (The *records* parameter is a signed value; if it is positive, the new USL is longer than the old USL; if it is negative, the new USL is shorter than the old USL.) The old USL file is copied to the new file with the same file name, and the old file is then deleted.

This intrinsic returns (as the value of EXPANDUSLF) the new file number. If an error occurs during the execution of EXPANDUSLF, the intrinsic instead returns an error number as its value.

The parameters are

uslfnm A word identifier supplying the filename of the USL file.

records An integer specifying the number of records by which the length of the USL file is to be changed. If less than 0, the file is contracted; if greater than 0, the file is expanded.

The condition codes possible are

CCE Request granted; the new file number is returned.

CCG (Not returned by this intrinsic.)

CCL Request not granted; an error number is returned. (The error numbers possible are discussed below.)

USL File Intrinsic Error Numbers

The INITUSLF, ADJUSTUSLF, and EXPANDUSLF intrinsics may return any of the following error numbers:

| Error Number | Meaning |
|--------------|----------------------------------------------------------------------------------------|
| 0 | Unexpected end-of-file was encountered. |
| 1 | Unexpected input/output error occurred. |
| 2 | An invalid <i>filecode</i> was specified. |
| 3 | An illegal file length was specified. |
| 4 | The user's request attempted to exceed the maximum file directory size (32,768 words). |
| 5 | Insufficient space was available in the USL file directory block. |
| 6 | Insufficient space was available in the USL file information block. |
| 7 | The utility was unable to open a new USL file. |
| 8 | The utility was unable to close (purge) an old USL file. |

| Error Number | Meaning |
|--------------|---------------------------------------------------------|
| 9 | The utility was unable to close (purge) a new USL file. |
| 10 | The utility was unable to close \$NEWPASS. |
| 11 | The utility was unable to close \$OLDPASS. |

CHANGING TERMINAL CHARACTERISTICS

Various commands and intrinsics can be issued against terminals to alter certain aspects of their operation. Before any of these intrinsics is issued against a terminal, the terminal/file must be opened with the FOPEN intrinsic.

On the HP 3000 Computer, terminals may be supported through either of the following controllers:

- Clock-Teletype Interface (Each controls one terminal.)
- Terminal Controller (Each controls up to 16 terminals.)

The Terminal Controller supports 103A and 202A modems, and hardwired terminals. The Clock-Teletype Interface supports only hardwired terminals.

Types of Terminals

The following user terminals can be connected to an HP 3000 Computer running under MPE/3000:

- HP 2600A or DATAPOINT 3300 Keyboard — Display Terminal (10/15/30/60/120/240 cps). Connectable through clock-teletype interface.
- ASR-33 EIA-compatible (HP 2749B) Terminal (10 cps). Connectable through clock-teletype interface.
- ASR-35 EIA-compatible Terminal (10 cps). Connectable through clock-teletype interface.
- General Electric TermiNet 300 Data Communications Terminal, Model B (10/15/30 cps) with Paper Tape Reader/Punch, Option 2. Connectable through clock-teletype interface.

NOTE: This terminal must be equipped for "ECHO PLEX."

- Memorex 1240 Communication Terminal (10/15/30/60/120/240 cps). *Not* connectable through clock-teletype interface.

NOTE: This terminal must be equipped with the even parity checking option.

- Execuport 300 Data Communications Transceiver Terminal (10/15/30 cps). *Not* connectable through clock-teletype interface.
- ASR-37 Teleprinter Terminal with Paper Tape Reader/Punch (15 cps). *Not* connectable through clock-teletype interface.
- IBM 2741 Communication Terminal (14.8 cps). (Call 360 Correspondence Code or PTTC/EBCD Code.) *Not* connectable through clock-teletype interface.

NOTE: This terminal must be equipped with the following features:

1. *Interrupt, Receive (IBM #4708) and Transmit (IBM #7900) associated with the terminal's ATTN key. (Terminals without Transmit (IBM #7900) are disconnected when log-on is attempted.)*
2. *Dial-Up (IBM #3255) to enable system connection through a 103A modem or acoustic coupler.*

- HP 2615A Terminal (10/15/30/60/120/240 cps).
- HP 2640A Interactive Display Terminal (Character mode only) (10-240 cps).

Terminals equipped with the automatic linefeed feature (operator selectable) must be operated with this feature OFF.

Special Keys

The following keys have special significance to MPE/3000:

| KEY | MEANING (MOST TERMINALS) | EXCEPTIONS |
|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| X ^c | Deletes (ignores) the line being typed and then reads any following characters. The system responds with a triple exclamation point (!!!) followed by a carriage-return and line-feed. (The superscript ^c denotes a control character. Thus, "X ^c " means "CONTROL-X.") | On IBM 2741, the user enters a two-character code followed by the character X. (Several ASCII characters and functions must be simulated by special two-character codes. These codes are described in Figure 8-3.) <i>pg 8-19</i> |
| H ^c | Deletes the previous character. (To delete <i>n</i> characters, enter <i>n</i> H ^c 's.) See note below. | On IBM 2741, the user enters a two-character code followed by the character H. |
| Q ^c | Places terminal in tape mode; input is from paper tape containing H ^c or X ^c 's. (Tape mode is not required to read tapes without these characters.) | |
| Y ^c | If the terminal is not in tape mode, Y ^c requests subsystem break (terminating program or command execution.) If the terminal is in tape mode, Y ^c returns it to the keyboard mode. | On an IBM 2741, a subsystem break is requested by the character sequence "ATTN Y" when not in input mode. (In input mode, this function is requested by "¢CY.") |
| BREAK | Requests a system break. | On an IBM 2741, a system break is requested by the sequence "ATTN ATTN." |
| ESC: | Places the terminal in the echo-on mode so that characters input are echoed to the terminal printer by MPE/3000. | All terminals except the IBM 2741 are assumed to be in <i>echo-on</i> mode when the user logs on. |
| ESC; | Places the terminal in echo-off mode so that characters input are not echoed by MPE/3000. | |
| LINE FEED | For any terminal with a line-feed key, the logon user may strike this key and a carriage return will be echoed. The line feed character is not transmitted to the input buffer. This mechanism permits multiple lines to be entered in response to a single read request (i.e., the length of a read request from a terminal is not constrained by the carriage or line width). | |

The defined control characters X^c , H^c , Q^c , and Y^c are recognized even when following 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 read as a two-character string in the user's input stream.

NOTE: The line correction mechanism (H^c) works in the following ways for all terminals except the IBM 2741 and the system console:

- *CRT Terminals*

All currently supported CRT's can physically backspace the cursor, therefore, H^c will cause the cursor to be backspaced one position, leaving the cursor positioned over the character to be replaced. The physical backspacing of the cursor does not erase the character from the screen, but the character has been deleted from MPE's internal buffer.

- *Hardcopy Terminals*

- a. *Terminals which have physical backspace capability: H^c causes a physical backspace to occur. In addition, a line feed is emitted unless the previous character was also a H^c . The result is that the user begins typing beneath the first character to be replaced.*

- b. *Terminals which have no physical backspace capability: No backspacing takes place. Each CONTROL-H will echo a backslash unless in tape mode.*

To take advantage of the backspacing (H^c) features, users must specify the appropriate TERM parameter at logon time. If the TERM parameter is omitted, the default is type zero (TTY) which will echo backslashes for each H^c .

IBM 2741 Communication Terminal Interface

Because the IBM 2741 terminal generates non-ASCII code, special consideration must be given to the representation of several ASCII characters and functions which are not available in the 2741 character set.

For input from a 2741 terminal, these characters (and some of the functions) are simulated by entry of a two-character code. The first character of this code is the cent symbol (¢). The cent symbol is followed by one of several alphanumeric or special characters (shown in Figure 8-3) to compose a unique code representing one ASCII character or function.

On input from a 2741 terminal, the two-character code is translated into the internal ASCII code. On output to a 2741 terminal, ASCII code is translated into the appropriate two-character representation.

The IBM 2741 Communication Terminal must be equipped with the interrupt feature associated with the *ATTN* key. This key represents the *break* function; it is used to terminate program or command execution.

Any CALL/360 or PTTC/EBCD characters that do not have an equivalent ASCII character are ignored on input.

Figure 8-3 shows 2741 terminal representation of ASCII characters and functions.

Changing Terminal Speed

MPE/3000 supports interactive terminals that run at speeds ranging from 10 to 240 characters per second (CPS). Once the user has logged on, he can change these speeds without logging-off by using the `:SPEED` command or the `FCONTROL` intrinsic (introduced for other functions in Section VI). This ability also allows a user running a mark-sense card-reader coupled to a terminal to operate the two devices at different speeds (for example, the card reader at 240 cps for input and the terminal at 10 cps for output). The `:SPEED` command and `FCONTROL` intrinsic are not valid for IBM 2741 Terminals, or other terminals that operate at only one speed, and do not apply to terminals connected through clock-teletype interfaces.

The `:SPEED` command can change both the input and output speed of the terminal. Its format is

$$\text{:SPEED} \left\{ \begin{array}{l} [\textit{inspeed}], \textit{outspeed} \\ \textit{inspeed} \end{array} \right\}$$

The following character sequences are defined to enable users of 2741 terminals to both input and output characters not available on the 2741 terminal.

| ASCII Terminal | | 2741 |
|----------------|-------------------|-----------------|
| < ¹ | Less Than | ¢L |
| > ¹ | Greater Than | ¢G |
| [| Left Bracket | ¢(|
| \ | Reverse Slant | ¢/ |
|] | Right Bracket | ¢) |
| ˆ | Circumflex | ¢A |
| ˘ | Grave Accent | ¢' |
| { | Opening Brace | ¢O |
| ¹ | Vertical Line | ¢V |
| } | Closing Brace | ¢S |
| ˜ | Tilde | ¢T |
| <i>DEL</i> | Delete | ¢D |
| <i>ESC</i> | Escape | ¢E |
| <i>BREAK</i> | System break | <i>ATTN</i> |
| <i>CONTROL</i> | Control character | ¢C |
| — ² | Underline | <i>UPSHIFT—</i> |

¹ Available on PTTC/EBCD terminal.

² Exists on some type balls.

Figure 8-3. ASCII vs 2741 Character Representation

inspeed The new input speed desired. (Required parameter if *outspeed* is omitted.)

outspeed The new output speed desired. (Optional parameter.)

Both of these values can be any of the following integers, specifying speed in characters per second.

10, 14, 15, 30, 60, 120, 240.

The following message is always output (at the old speed):

CHANGE SPEED AND INPUT "MPE":

In response, the user manually changes the speed control on the terminal and verifies the change by entering the message:

MPE

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

Through the FCONTROL intrinsic, the user may change the terminal input or output speed programmatically. The format of this intrinsic is

PROCEDURE *FCONTROL (filenum,controlcode,speed);*

VALUE *filenum,controlcode;*

INTEGER *filenum,controlcode;*

LOGICAL *speed;*

OPTION EXTERNAL;

The FCONTROL parameters are

filenum A word identifier supplying the filenumber of the terminal for which the speed is changed.

controlcode The integer 10 (decimal) to change the input speed or the integer 11 to change the output speed.

speed A reference parameter or word identifier that specifies the new speed desired: 10, 14, 15, 30, 60, 120, or 240 characters/second. When the FCONTROL intrinsic is executed, the previous input or output speed is returned to the calling process through the *speed* parameter.

The condition codes are

- CCE The request was granted.
- CCG (Not returned.)
- CCL The request was not granted; the process does not own the logical device, or this device is not a terminal, or the speed entered is not acceptable.

EXAMPLE:

To change the current input speed of the terminal identified by the filenumber stored in the word TERMFN from 60 to 120 characters per second, the user issues the following call. (The word SPD contains the value 120.)

FCONTROL (TERMFN, 10, SPD);

After the intrinsic is executed, the word SPD contains the integer 60 (the previous speed).

Changing Input Echo Facility

Users can programmatically determine whether MPE/3000 transmits (echoes) input from the terminal keyboard back to the terminal printer, by calling the FCONTROL intrinsic to turn the echo facility on or off.

When the echo facility is *on*, input read from the terminal is echoed to the terminal's printer by MPE/3000. 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, however, the echoed printing is illegible — as the user enters input on such terminals, it is simultaneously printed by the terminal itself and subsequently overwritten by the echoed information. (Where a terminal can operate in either full- or half-duplex mode, the mode is selected by a switch on the terminal. When the user logs on, all terminals except 2741 terminals are assumed to have the echo facility *on*.)

When the echo facility is *off*, input read from the terminal is not echoed to the terminal's printer by MPE/3000. 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 8-4.)

In addition to the FCONTROL intrinsic, the echo facility also can be switched on and off by entering the characters:

- ESC:* to turn echo facility *on*
- ESC;* to turn echo facility *off*.

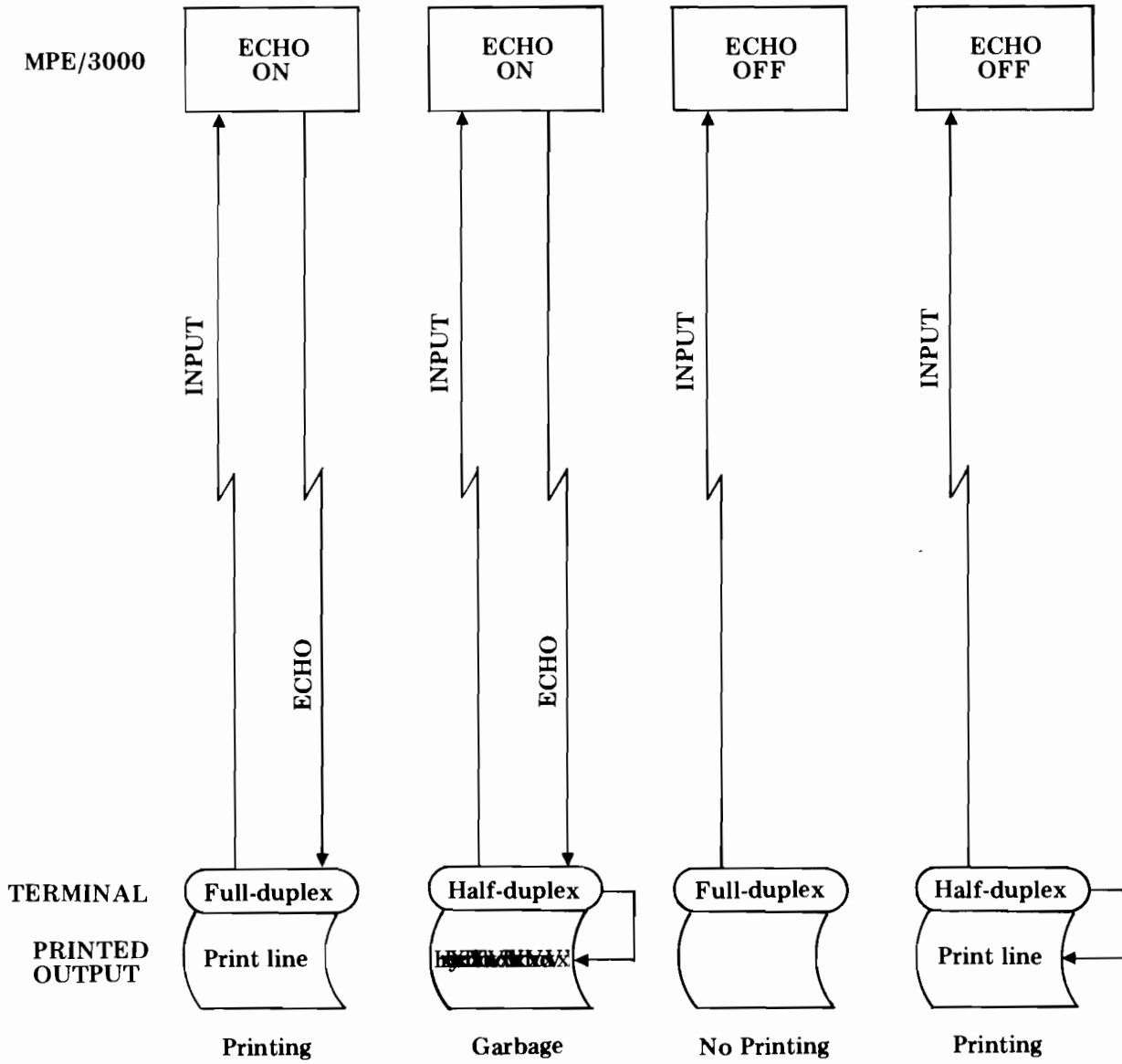


Figure 8-4. Echo Facility vs Duplex Mode

The format for this application of the FCONTROL intrinsic is

```
PROCEDURE FCONTROL (filenum,controlcode,last);  
VALUE filenum,controlcode;  
INTEGER filenum,controlcode;  
LOGICAL last;  
OPTION EXTERNAL;
```

The FCONTROL parameters are

| | |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------|
| <i>filenum</i> | A word identifier supplying the filenumber of the user's terminal. |
| <i>controlcode</i> | The integer 12 to turn the echo facility <i>on</i> , or the integer 13 to turn it <i>off</i> . |
| <i>last</i> | A reference parameter to which the <i>previous</i> echo facility is returned, where: 0 = Echo on. 1 = Echo off. |

The condition codes are

| | |
|-----|-------------------------------------------------------------------------------------------------------|
| CCE | The request was granted. |
| CCG | (Not returned by this intrinsic.) |
| CCL | The request was not granted; the file specified did not belong to this process or was not a terminal. |

Enabling and Disabling System Break Function

The user can programmatically suspend or enable a terminal's ability to react to a system break request by calling the FCONTROL intrinsic. (System break requests are initialized by pressing the *break* key or by calling the CAUSEBREAK intrinsic.)

PROCEDURE *FCONTROL (filenum,controlcode,anyinfo);*

VALUE filenum,controlcode;

INTEGER filenum,controlcode;

LOGICAL anyinfo;

OPTION EXTERNAL;

The FCONTROL parameters are

- filenum* A word identifier supplying the filenumber of the terminal for which the break function is enabled or disabled.
- controlcode* The integer 15 to enable the break function, or the integer 14 to disable the break function.
- anyinfo* Any variable or word identifier. This parameter is needed by FCONTROL to satisfy the internal requirements of this intrinsic. However, this parameter serves no other purpose and is not modified by the intrinsic.

The condition codes are

- CCE The request was granted.
- CCG (Not returned by this intrinsic.)
- CCL The request was not granted, because the file number specified does not denote a terminal or does not belong to the calling process.

Enabling and Disabling Subsystem Break Function

All terminals are initially set to disable (not accept) subsystem break requests, generated by pressing the *CONTROL-Y* (Y^c) key during a session. But the user can programmatically enable and again disable a terminal's ability to react to subsystem break requests by calling the FCONTROL intrinsic.

PROCEDURE *FCONTROL (filenum,controlcode,anyinfo);*

VALUE filenum,controlcode;

INTEGER filenum,controlcode;

LOGICAL anyinfo;

OPTION EXTERNAL;

The FCONTROL parameters are

| | |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>filenum</i> | A word identifier supplying the filenumber of the terminal for which the subsystem break function is enabled or disabled. |
| <i>controlcode</i> | The integer 17 to enable the escape function, or the integer 16 to disable the subsystem break function. |
| <i>anyinfo</i> | Any variable or word identifier. This parameter is needed to satisfy the internal requirements of the intrinsic. However, this parameter serves no other purpose and is not modified by the intrinsic. |

The condition codes are

| | |
|-----|--------------------------------------------------------------------------------------------------------------------------------------|
| CCE | The request was granted. |
| CCG | (Not returned by this intrinsic.) |
| CCL | The request was not granted, because the file number specified does not denote a terminal or does not belong to the calling process. |

Enabling and Disabling Parity-Checking

All terminals and mark-sense card readers are initially set to disable parity-checking during read operations. They may, however, be programmatically enabled for parity-checking by calling the FCONTROL intrinsic. Then, the parity of the data received is checked against the parity computed by the HP 3000 Asynchronous Channel multiplexor. If a parity error is detected, an error code is made available to the user, who obtains it through the FCHECK intrinsic. When a user is running a card reader coupled to a terminal, the ability to enable/disable parity-checking allows him to obtain optimum utilization of the card reader by running it at 240 characters/second without printing characters, while still outputting characters for the terminal.

PROCEDURE *FCONTROL (filenum,controlcode,anyinfo);*

VALUE *filenum,controlcode;*

INTEGER *filenum,controlcode;*

LOGICAL *anyinfo;*

OPTION EXTERNAL;

The FCONTROL parameters are

- filenum* A word identifier supplying the filenumber of the terminal for which the parity check function is enabled or disabled.
- controlcode* The integer 24 to enable parity-checking, or the integer 23 to disable parity-checking.
- anyinfo* Any variable or word identifier. This parameter is needed to satisfy the internal requirements of the intrinsic. However, this parameter serves no other purpose and is not modified by the intrinsic.

The condition codes are

- CCE The request was granted.
- CCG (Not returned by this intrinsic.)
- CCL The request was not granted because the file number specified does not denote a terminal, or does not belong to the calling process.

Enabling and Disabling Tape-Mode Option

The user can programmatically enable or disable the tape-mode option for a terminal. When enabled, the tape-mode option inhibits the implicit line-feed normally issued by MPE/3000 each time a carriage-return is entered. The tape-mode option also inhibits responses to H^c and X^c entries. (Thus, when H^c is received and tape-mode is enabled, no reverse slash (\) is sent to the terminal; when X^c is received and tape-mode is in effect, no exclamation points (!!!) are sent to the terminal.) The tape-mode option is enabled or disabled with the FCONTROL intrinsic.

```
PROCEDURE FCONTROL (filenum,controlcode,anyinfo);
```

```
VALUE filenum,controlcode;
```

```
INTEGER filenum,controlcode;
```

```
LOGICAL anyinfo;
```

```
OPTION EXTERNAL;
```

The FCONTROL parameters are

| | |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>filenum</i> | A word identifier supplying the filenumber of the terminal for which the tape-mode is enabled or disabled. |
| <i>controlcode</i> | The integer 19 to enable tape-mode, or the integer 18 to disable tape-mode. |
| <i>anyinfo</i> | Any variable or word identifier. This parameter is needed to satisfy the internal requirements of the intrinsic. However, this parameter serves no other purpose and is not modified by the intrinsic. |

The condition codes are

| | |
|-----|--------------------------------------------------------------------------------------------------------------------------------------------------|
| CCE | The request was granted. |
| CCG | (Not returned by this intrinsic.) |
| CCL | The request was not granted because the filenumber specified does not denote a terminal, or the terminal does not belong to the calling process. |

Reading Paper Tapes Without X-OFF Control

The *X-OFF* control character (written by pressing the *X-OFF* key on a teletype) is used to delimit data input on paper tape. When a TTY Tape Reader encounters this character while reading a tape, reading halts until the program requests more input data.

If a user is working at a terminal with a paper tape reader that does not recognize the *X-OFF* character, he can input data from paper tape by using the `:PTAPE` command. (This command strips all *X-OFF* characters from the input read.) The input terminates when the Y^c (*CONTROL-Y*) character is encountered, returning control to the user at the terminal keyboard. The `:PTAPE` command is written as follows:

`:PTAPE filename`

| | |
|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>filename</i> | The name of an existing ASCII file on disc, to which the input data is written. (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.) |
|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

The `:PTAPE` command (and `PTAPE` intrinsic) cannot be used to copy tapes containing more than 24576 words (49152 bytes).

The user can programmatically read data from paper tapes not containing the *X-OFF* control character, or from tapes input through terminals not recognizing this character, by calling the `PTAPE` intrinsic. (The *X-OFF* characters are stripped from the tape.) Tape input terminates when Y^c is encountered, returning control to the user at the terminal keyboard. (Prior to calling this intrinsic, the user must be sure to position the end-of-file pointer to the proper position in the file. If he is inputting more than one tape, he should specify, in the `FOPEN` call that opens the file, the *append-only* access type, and a *variable-length* record format, before the first `PTAPE` call; furthermore, he should set the end-of-file pointer to zero, if necessary, before issuing the first `PTAPE` call.)

Lines will be folded at 256-character intervals until a carriage control character indicates the end of a line or until the input is terminated by the Y^C character.

The PTAPE intrinsic format is

```
PROCEDURE PTAPE (filenum1,filenum2);  
VALUE filenum1,filenum2;  
INTEGER filenum1,filenum2;  
OPTION EXTERNAL;
```

The intrinsic parameters are

| | |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>filenum1</i> | A word identifier supplying the filenumber of the user's terminal. (This is the value returned by FOPEN when this terminal/file is opened.) |
| <i>filenum2</i> | A word identifier supplying the filenumber of the disc file to which the data is to be written. (This is the value returned by FOPEN when this disc file is opened.) |

The PTAPE intrinsic can return the following condition codes:

| | |
|-----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CCE | Request granted. |
| CCG | Request not granted because an error occurred while writing to the specified disc file. |
| CCL | Request not granted because the input file specified is not a terminal or does not belong to the calling process, or because insufficient resources (such as disc space or main memory) are available to satisfy the request. |

Enabling and Disabling the Terminal Input Timer

The terminal input timer records the time required to satisfy an input request on the terminal, from the time the input is requested, until it is completed. (This applies only to unbuffered, serial terminal input requests.) The user can programmatically enable or disable the terminal timer by calling the FCONTROL intrinsic.

```
PROCEDURE FCONTROL (filenum,controlcode,anyinfo);  
VALUE filenum,controlcode;  
INTEGER filenum,controlcode;  
LOGICAL anyinfo;  
OPTION EXTERNAL;
```

The FCONTROL parameters are

- filenum* A word identifier supplying the filenumber of the terminal for which the timer is enabled or disabled.
- controlcode* The integer 21 to enable the timer, or the integer 20 to disable the timer.
- anyinfo* Any variable or word identifier. This parameter is needed to satisfy the internal requirements of the intrinsic. However, this parameter serves no other purpose and is not modified by the intrinsic.

The condition codes are

- CCE The request was granted.
- CCG (Not returned by this intrinsic.)
- CCL The request was not granted because the filenumber specified does not denote a terminal, or the terminal does not belong to the calling process.

Reading the Terminal Input Timer

The user can read the result from the terminal input timer through the FCONTROL intrinsic. This result will be valid only if the terminal input was preceded by a call to enable the terminal timer. If valid, the result is the time (in hundredths of seconds) required for the last direct, unbuffered serial input on the terminal specified.

```
PROCEDURE FCONTROL (filenum,controlcode,inputtime);  
  
VALUE filenum,controlcode;  
  
INTEGER filenum,controlcode;  
  
LOGICAL inputtime;  
  
OPTION EXTERNAL;
```

The FCONTROL parameters are

- filenum* A word identifier supplying the filenumber of the terminal for which the timer is read.

controlcode The integer 22.

inputtime A word to which is returned the input time (in seconds/100).

The condition codes are

CCE The request was granted.

CCG The request was granted, but the result overflow 16-bits (*inputtime* was greater than 655.35 seconds).

CCL The request was not granted because the filenumber specified does not denote a terminal or the terminal does not belong to the user's process.

Defining Line-Termination Characters For Terminal Input

Normally, the user at a terminal indicates the end of a line of input by entering a carriage return (through the *return* key on most terminals). He can, however, specify that a different character (such as an equal sign, a period, or exclamation point) be recognized as the standard line terminator during his process. On subsequent read operations, the input line will be terminated upon receipt of the specified character. That character will be returned to the requester's buffer. No line-feed or carriage-return will be generated.

The line-terminator character change is requested through the FCONTROL intrinsic.

PROCEDURE *FCONTROL (filenum,controlcode,character);*

VALUE *filenum,controlcode;*

INTEGER *filenum,controlcode;*

LOGICAL *character;*

OPTION EXTERNAL;

The FCONTROL parameters are

filenum A word identifier supplying the filenumber of the terminal to which the new terminating character applies. (This parameter *must* specify a terminal.)

controlcode The integer 25.

character A word identifier supplying (in the right byte) the character to be used as a line terminator. (The left byte of this word can contain any information--it is ignored by the intrinsic.)

The condition codes are

- CCE The request was granted.
- CCG (Not returned by this intrinsic.)
- CCL The request was not granted; the character specified is disallowed.

The following characters are *not* allowed as line-terminating characters in the *character* parameter.

| ASCII Character | Octal Code |
|-----------------------------|---------------|
| Control Y (Y ^c) | 31 |
| Carriage-return | 15 |
| Line-feed | 12 |
| ESC | 33 |
| Control X (X ^c) | 30 |
| Control H (H ^c) | 10 |
| Control Q (Q ^c) | 21 |
| X-OFF | 23 |

If the character supplied equals zero, the terminal will revert to normal line-control operation.

DEVICE CHARACTERISTICS

Paper Tape Reader

The paper tape reader driver is capable of reading tapes in either **BINARY** or **ASCII** format. The mode is determined by the **ASCII/BINARY** bit **FOPTIONS** word of the **FOPEN** intrinsic. With the **FCONTROL** intrinsic, the default condition of **ASCII** mode may be altered.

BINARY MODE. In binary mode, the device will read the number of words/bytes requested without regard to any special characters present on the tape. Tape leaders are skipped automatically by the hardware; (i.e., whenever the roller is raised to load a new tape, the device sets an internal flag which causes it to ignore all leading null characters on the next read). After this first read, the flag is reset. Thus, trailers and imbedded nulls are as valid as any other characters. The full 8 bit character is read, and characters are packed two characters per word. The end of record character is not recognized, so the read is terminated only when the word/byte count is satisfied.

If a time-out occurs, (i.e., for a tape bind, broken tape, or an attempt to read beyond the end of the tape), an error (CCL) is returned to the calling program.

ASCII MODE. In ASCII mode, the following conditions apply by default:

- Bit 8 (parity bit) is set to zero.
- The Carriage-return character (%15) is recognized as the end-of-record character. In other words, data transfer stops when the word/byte count is satisfied or when the end-of-record character is detected. (However, if word/byte count is satisfied before the end-of-record character is found, the tape motion continues until end-of-record is detected, with the extra characters being ignored.)
- Data characters are packed automatically with two characters per computer word.

The following characters are not transferred as data and result in the following action:

| | |
|-----------------------|------------------------------------------------------------------------------------------------------------|
| Carriage-return (%15) | End-of-record. |
| Line-feed (%12) | Ignored. |
| X-ON (%21) | Ignored. |
| X-OFF (%23) | Ignored. |
| Rub-out (%177) | Ignored. |
| Control H (%10) | Previous character deleted from data buffer. |
| Control X (%30) | All data in current record are deleted. The tape is advanced to the next record and the read is restarted. |
| Control Y (%31) | Ignored. |
| Null (% 0) | Consecutive nulls are counted to determine end-of-tape. |

- Any number of leading Null characters are allowed. However, after the first non-zero character in a record, 20 consecutive Null characters will be treated as the end-of-tape condition and will result in the following actions:
 - a. Tape motion is stopped.
 - b. Any characters already read are deleted.
 - c. The message

LDEV #nn NOT READY

is output to the system console. The operator should insert the next tape to be read into the paper tape reader.

- d. When the **READY** interrupt is detected, the **READ** request is restarted and operation continues as before.

The entire sequence is transparent to the calling program, (except for the delay required to change tapes), so that multiple tapes may be read as if they were a single tape.

NOTE: There should always be at least one non-Null character (such as a Line-feed) before the TRAILER to allow it to be distinguished from the LEADER.

- All Job Control Cards, (i.e. :JOB, :DATA, :EOD, :EOJ, etc.), are recognized in the standard way to allow batch input from paper tape.
- When the paper tape reader file is OPEN, its characteristics may be altered with FCONTROL (refer to section on MPE INTRINSICS). (However, when the device is closed, all parameters revert back to the default state.

MPE INTRINSICS. The paper tape reader driver is compatible with all MPE intrinsics. The device dependent FCONTROL call, i.e. FCONTROL (*filenum, 0, param*), has the following meaning:

param < 0

Change the end-of-tape sensing. By default, when 20 consecutive Null characters are read, the driver will stop reading and issue the NOT READY message to the system console. (If the user wishes to change this feature, he should call FCONTROL with the value of *param* equal to the negative of the desired maximum number of consecutive Null characters.)

$0 \leq \textit{param} \leq \%177$

Change end-of-record sensing. By default, the record terminator is a Carriage-return (%15). If the user wishes to change the record termination character, he should call FCONTROL with the value of the desired record termination character.

param = %200

Set ASCII mode. If the user wishes to set ASCII mode after the file has been opened, he should call FCONTROL with the value of *param* equal to %200. The maximum Null count and record-termination character are unaffected by this call.

param = %201

Set BINARY mode. If the user wishes to set BINARY mode after the file has been opened, he should call FCONTROL with the value of *param* equal to %201. The values of the maximum Null count and record-terminator are unaffected by this call. (Of course, the record-termination character and maximum Null count have no meaning in BINARY. If the user switches between BINARY and ASCII mode, they will remain unchanged.)

$\%202 \leq param \leq \%77777$ Reserved. Do not use these values of *param*.
At present they will be treated as *param* modulo
 $\%200$, but this may change later.

NOTE: All parameters return to their default value when the device is closed.

Paper Tape Punch

The paper tape punch driver is capable of punching tapes in either BINARY or ASCII format. The mode is determined by the ASCII/BINARY bit in FOPTIONS word of the FOPEN intrinsic. With the FCONTROL intrinsic, the default condition of ASCII mode may be altered.

BINARY MODE. In binary mode, all characters are punched exactly as they appear in the buffer. No characters are deleted or added. Data characters are automatically unpacked, assuming two characters per computer word. In other words, no end of record marks such as CR are punched unless they are in the buffer. All bit patterns are considered valid and none have any significance as far as the driver is concerned. If the user wants end-of-record marks on the tape, he must provide them himself. (Note, though, that the paper tape reader driver also attaches no significance to the end-of-record marks.)

ASCII MODE. In ASCII mode, the following conditions apply by default:

- Trailing blank characters ($\%40$) are not output to the tape. This feature saves paper tape on short records, and also speeds up the net transfer rate (for output and for input when the tape is read.)
- All other characters, including leading and imbedded blanks, are transferred as data. No special characters are recognized.
- A record-termination sequence consisting of a Carriage-return ($\%15$) followed by a Line-feed ($\%12$), is appended to the end of each record.
- Data characters are automatically unpacked, assuming two characters per computer word.

MPE INTRINSICS. The paper tape punch driver is compatible with all MPE intrinsics. The device dependent FCONTROL call, i.e. FCONTROL (*filenum*, 0, *param*), has the following meaning:

| | |
|------------------------|---------------------------------------------------------------------------------------|
| <i>param</i> = 0 | Request the operator to mount a new roll of tape. |
| <i>param</i> = $\%200$ | Set ASCII mode (refer to MPE INTRINSIC section on Paper Tape Reader). |
| <i>param</i> = $\%201$ | Set BINARY mode (refer to MPE INTRINSIC section on Paper Tape Reader). |
| <i>param</i> = $\%202$ | Prepare to change the record-termination sequence. By default, the record-termination |

sequence is Carriage-return followed by Line-feed. If the user wishes to modify this sequence he should call FCONTROL with the value of *param* equal to %202 and then call FWRITE with the desired record-termination sequence in the data buffer. This record-termination sequence may be up to four characters in length. All characters in any sequence are valid.

NOTE: All parameters are returned to the default value when the device is closed.

LEADER/TRAILER: When the device is opened, *nn* consecutive Null characters are output to serve as LEADER. Similarly, when the device is closed, *nn* Null characters are output to serve as TRAILER.

Card Reader

The card reader is a unit record device. The data is read in packed ASCII mode; that is, two columns are converted to ASCII and packed into the left and right byte of one word. If the read request specifies 80 or more bytes, 80 bytes will be transmitted independent of the data on the card.

Line Printer

The line printer is a print and space device; i.e., post space. The prespace operation is simulated by doing a print operation and then filling the line printer buffer.

The following table describes the differences between the three subtypes of line printers:

| SUBTYPE | HP PRODUCT NO. | SUPPRESS SPACE | VERTICAL SPACING | CHANNELS 9-12 |
|---------|----------------|----------------|------------------|---------------|
| 0 | 30108A | YES | 63 | NO*** |
| 0 | 30109A | YES | 63 | NO*** |
| 1 | 30118A | NO* | 15** | NO*** |
| 2 | 30127A | YES | 15** | YES |
| 2 | 30128A | YES | 15** | YES |

*A suppress space request to subtype 1 will result in a single space.

**Carriage control codes %220-%277 will space 15 lines on subtypes 1 and 2.

***A request to skip to channels 9-12 (carriage control codes %310-%313) will result in a single space on subtypes 0 and 1.

The HP 30108A/30109A printers use a 6-bit space count that allows vertical spacing of up to 63 lines when carriage control codes of %200-%277 are used.

The HP 30108A/30109A/30118A printers have only a 4-bit space count that imposes a maximum vertical spacing limit of 15 lines. MPE/3000 will not simulate vertical spacing of more than 15 lines for these printers. Carriage control codes %217-%277 will space 15 lines on these printers. User programs and subsystems like RPG/3000, which permit vertical spacing of up to 112 lines, should always perform vertical spacing of more than 15 lines by issuing successive 15-line spaces. This technique will permit such programs to be device-independent; also, it is impossible for such programs to accurately determine the printer subtype under certain conditions when the file is spooled (see devtype parameter of FGETINFO).

Magnetic Tape

The magnetic tape unit reads and writes variable length records in a packed binary mode. Each word of data is represented by two tape characters. On read requests, the amount of data transferred is the lesser of the read request length and the tape record length.

After write operations, when the end of tape reflective marker is detected, an EOT indication is returned to the user. A request which was initiated before the EOT marker was detected is completed but an EOT indication is returned.

Card Punch

The HP 30112A Card Punch operates, by default, in ASCII mode. However, when the device is not spooled, its mode can be changed by issuing the following intrinsic:

PROCEDURE

FCONTROL(filenum,0,param)

where *param* = 0 selects column binary mode
 = 1 selects ASCII mode
 = 2 selects packed binary mode

Printing Reader/Punch

The HP 30119A Printing Reader/Punch is supported in three ways by MPE:

1. As a card reader which, from a user program's viewpoint, behaves exactly like the HP 30106A/30107A card readers. This mode of operation prevails when the device is opened by device class name and the device class access type is input-only.

2. As a card punch which, from a user program's viewpoint, behaves exactly like the HP 30112A card punch. This mode of operation prevails when the device is opened by device class name and the device class access type is output-only.
3. As a printing reader/punch with two input hoppers and two output stackers over which the user has complete control. This mode of operation prevails when the device is opened by logical device number or by device class name when the device class access type is input/output.

In the latter mode of operation, it is the user's responsibility to control all aspects of the device by using the following intrinsic:

PROCEDURE

| |
|-----------------------------------------|
| <i>FCONTROL (filenumber, 0, param);</i> |
|-----------------------------------------|

| | | |
|--------------------------|---|----------------------------------------------------|
| where <i>param</i> (0:5) | = | (reserved) |
| <i>param</i> (5:1) | = | 0 — no action |
| | = | 1 — feed one card |
| <i>param</i> (6:1) | = | 0 — select no inhibit feed on writes |
| | = | 1 — select inhibit feed on writes |
| <i>param</i> (7:1) | = | 0 — select punch, on writes |
| | = | 1 — select no punch on writes |
| <i>param</i> (8:1) | = | 0 — select print on writes |
| | = | 1 — select no print on writes |
| <i>param</i> (9:1) | = | 0 — select print and punch same data on writes |
| | = | 1 — select print and punch separate data on writes |
| <i>param</i> (10:1) | = | 0 — select primary stacker |
| | = | 1 — select secondary stacker |
| <i>param</i> (11:1) | = | 0 — select primary hopper |
| | = | 1 — select secondary hopper |
| <i>param</i> (12:2) | = | 0 — select column binary mode |
| | = | 1 — select ASCII mode |
| | = | 2 — select packed binary mode |
| | = | 3 — (invalid) |
| <i>param</i> (14:2) | = | (reserved) |

When the device is opened for the first time, all of the above parameter selections assume a default value of zero except for the mode, which is ASCII. However, real locations of the device via subsequent opens will not necessarily yield these default values; the parameter selections for such opens will assume the values currently selected by previous opens.

The FREAD/FWRITE intrinsics perform the following actions:

- FREAD
- a. feeds a card from the hopper last selected;
 - b. moves card from wait station (if present) to stacker last selected (no punch or print performed);
 - c. reads data from (a) and puts it in caller's buffer;
 - d. the mode (ASCII, column binary, etc.) of the read will be the one last selected;
 - e. the following parameter selections have no effect:
 - same/separate print data;
 - print/no print;
 - punch/no punch;
 - inhibit input feed.
- FWRITE
- a. moves card from the wait station to the stacker last selected;
 - b. prints and/or punches card (1) using data in caller's buffer;
 - c. if inhibit input feed has been selected, no card is fed from hoppers; if inhibit input feed has not been selected, card will be fed from hopper last selected; any data on that card is lost.
 - d. if separate print data has been selected; a double buffer is expected and punch data will be extracted from the first part, print data from the second; no print and no punch selections will still be honored, however (if either one is on, a single length buffer will suffice); if separate print data has not been selected, then the same data is printed and punched, unless no print or no punch has been selected;
 - e. the mode (ASCII, column binary, etc.) used will be the one last selected;
 - f. if no print is selected, then printing is inhibited;
 - g. if no punch is selected, then punching is inhibited.

The FCONTROL parameter selections (bits 6-13) remain in force until changed by another FCONTROL. If bit 5 of the parameter word is on, one card will be fed after the other option selections are updated:

- a. card will be fed from the hopper selected;
- b. card in wait station will be moved to the stacker selected;
- c. no data will be read.

SECTION IX

Resource Management

Within MPE/3000, any element that can be accessed by a user's program is regarded as a *resource*. A resource thus can be an input/output device, file, program, subroutine, procedure, code segment, or the data stack. Occasionally, the user may want to manage a specific resource shared by a particular set of jobs or processes, so that no two of these jobs or processes can use the resource at the same time. To accomplish this type of resource management on either the inter-job (or session) or inter-process level, the jobs or processes involved must mutually cooperate. For example, if JOBB must not access a particular file when JOBA is using it, both jobs should include provisions for a hand-shaking arrangement overseen by MPE/3000 when these jobs are being executed concurrently. Under this arrangement, when JOBA has exclusive access to the file and JOBB attempts to access the same file, this access will be denied. JOBB will be suspended until JOBA releases its exclusive access. Then, JOBB can resume execution and access the file. (It is important to realize that as long as JOBB is suspended, it not only cannot access the file—it cannot perform *any* operations.)

On either the inter-job or inter-process level, the hand-shaking arrangement is based upon an arbitrary *resource identification number (RIN)* made available to users (at the inter-job level) or assigned to the job (at the inter-process level). Within their jobs (or processes), the cooperating programmers relate a RIN to a particular resource through the structure of the statements making up each job (or process). When a job (or process) seeks exclusive access to a resource, it requests MPE/3000 to lock the related RIN. (This request is granted only if no other job or process has already locked the RIN; otherwise, the requesting process is suspended until the RIN is released.) When it is finished with the resource, the job (or process) requests MPE/3000 to unlock the RIN so that other jobs (or processes) can lock it.

A RIN is *not* a *physical entity*. Furthermore, it is *not* logically *assigned* to any resource. The association between a RIN and a resource is accomplished only by the context of the statements within the job or process using the RIN. (This technique is illustrated later in this section.) The RIN is always known to MPE/3000 but its meaning (the resource with which it is associated) is not. For this reason, all cooperating programs must agree on what RIN is associated with what resource.

Processes run by users having only the *Standard MPE/3000 Capabilities* can lock only one RIN at a time. But processes run by users having the *Multiple RIN Optional Capability* discussed in Section XIII can lock more than one at a time. In doing so, however, the users must be careful to avoid deadlocking, where two suspended processes cannot be resumed because they are mutually blocked.

INTER-JOB LEVEL

The RIN's used at the inter-job level are called *global RIN's*. They are used to exclude simultaneous access of a resource by two or more jobs. Each global RIN is a positive integer unique within MPE/3000. Global RIN's are acquired and released through commands, and locked and unlocked through intrinsics.

Acquiring Global RIN's

Before any users can mutually engage in resource management through a RIN, one of these users must request the RIN and assign it a RIN password that enables all who know the password to lock the RIN. He does this by entering the :GETRIN command:

```
:GETRIN rinpassword
```

rinpassword A password that will be required in the intrinsic that locks the RIN. This is a string of up to eight alphameric characters beginning with a letter. (Required parameter.)

The :GETRIN command is typically entered during a session. As a result of this command, MPE/3000 makes a RIN available for use, and displays the RIN number in this format:

RIN: rin

rin The RIN number.

The user issuing :GETRIN can use this *rin* number to lock and unlock the RIN in the current session, or in future jobs and sessions. He also tells the RIN number (and password) to other users to permit them to lock and unlock the RIN in their jobs and sessions. All users pass the RIN number to the intrinsics that lock and unlock the RIN, as a reference parameter in the intrinsic calls. These users can continue using the RIN until the user who issued :GETRIN releases it. Thus, a user may use the same RIN for several days or weeks. (MPE/3000 regards the user issuing :GETRIN as the owner of the RIN assigned. This means that only this user may release the RIN.)

The total number of RINS that MPE/3000 can allocate is specified when the system is configured, and in no case can exceed 1024.

Locking and Unlocking Global RIN's

Any global RIN assigned to a group of users can be locked (by one job at a time) by issuing the LOCKGLORIN intrinsic call. When this is done, any other jobs that attempt to lock this RIN are suspended. The requestor must know both the RIN number and password. Users with only *Standard MPE/3000 capabilities* cannot lock more than one global RIN simultaneously; an attempt to do so aborts the job. The intrinsic used to lock a global RIN is

PROCEDURE

LOCKGLORIN (*rinum,lockcond,rinpassword*) ;

VALUE *rinum*;

LOGICAL *rinum,lockcond*;

BYTE ARRAY *rinpassword*;

OPTION EXTERNAL;

The parameters are

| | |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>rinum</i> | A word supplying the RIN number of the resource to be locked. (This is the RIN number furnished in the :GETRIN command.) |
| <i>lockcond</i> | A word specifying conditional or unconditional RIN locking, as follows: TRUE = Locking will take place unconditionally; if the RIN is not available, the calling process suspends until it is. (The TRUE condition is passed as a word whose 15th bit is <i>on</i> ; all other bits are ignored.) FALSE = Locking will take place only if the RIN is immediately available; if it is not, control returns to the calling process immediately, with the condition code CCG. (The FALSE condition is passed as a word whose 15th bit is <i>off</i> ; all other bits are ignored.) |
| <i>rinpassword</i> | A byte array containing the RIN password assigned through the :GETRIN command. In this array, the password must be terminated by an ASCII special character. |

The condition codes possible if *lockcond* = TRUE are

| | |
|-----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CCE | The request was granted. If the calling process had already locked the RIN, FALSE is returned to the word <i>lockcond</i> ; if the RIN was free, TRUE is returned to <i>lockcond</i> . |
| CCG | (Not returned.) |
| CCL | (Not returned.) |

This condition codes possible if *lockcond* = FALSE are

| | |
|-----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CCE | The request was granted. If the calling process had already locked the RIN, FALSE is returned to the word <i>lockcond</i> ; if the RIN was free, TRUE is returned to <i>lockcond</i> . |
| CCG | The request was not granted because the RIN was locked by another job. |
| CCL | (Not returned.) |

The LOCKGLORIN intrinsic is aborted if the process already has another global RIN locked (and the user does not have the Multiple RIN Optional Capability); if an incorrect password is used; or if the specified RIN is not assigned to the job.

To unlock the same RIN, the user calls the following intrinsic:

```
PROCEDURE      UNLOCKGLORIN  (rinum) ;
VALUE rinum;
LOGICAL rinum;
OPTION EXTERNAL;
```

In this call, the *rinum* parameter must be a word supplying the number of the last RIN locked by the job, since only one RIN per job can be locked at one time. If *rinum* does *not* specify this RIN, no action is taken. If no UNLOCKGLORIN intrinsic is issued for a locked RIN during a job, that RIN remains locked until :EOJ is encountered, then, it is unlocked.

The condition codes possible are

| | |
|-----|--------------------------------------------------------------------------------------|
| CCE | The request was granted. |
| CCG | The request was <i>not</i> granted because this RIN was not locked for this process. |
| CCL | The RIN was not allocated. |

To illustrate how the above two intrinsic calls are used, consider two jobs run by two users who cooperate in managing a common disc file named RES3. Although both jobs may alter the contents of RES3, the users want to avoid the problems that would occur if both jobs attempted to write on the file simultaneously. Assuming that the users are utilizing RIN 42, allocated through a :GETRIN command, they might manage the file RES3 as illustrated. (RIN 42 is passed to the RIN-management intrinsics in the word RINX. The password associated with RIN 42 is stored in the array RINPASS. The 15th bit of the word TRU is *on*.)

When the first FWRITE intrinsic call in JOBA references the file RES3, no LOCKGLORIN intrinsic call has yet been issued by this job or by JOBB, running concurrently. Thus, this FWRITE call can access RES3. Subsequently, JOBA issues a LOCKGLORIN call that locks RES3 (and any other resources referenced between this LOCKGLORIN call and the subsequent UNLOCKGLORIN call). While RES3 is locked, *any* job can access it in the normal fashion, but *none* can lock it for exclusive use. Because the programmer who wrote JOBB does not want to access RES3 while this file is locked by another job, he specifies a lock call before he references this file in a FWRITE call. (In other words, he tries to lock the file for JOBB.) When the LOCKGLORIN call in JOBB is encountered and JOBA still has RES3 locked, JOBB is suspended. As soon as JOBA unlocks RES3, however, JOBB resumes execution and writes on this file.

EXAMPLE:

```
:JOB,JOBA,ACCT.JOE
.
.
.

:SPL
.
.
.
FWRITE (RES3,...);
.
.
.
LOCKGLORIN (RINX,TRU,RINPASS);
.
.
.

FWRITE (RES3,...);
.
.
.

UNLOCKGLORIN (RINX);
.
.
.

:EOD
:EOJ
```

TIME



```
:JOB,JOBB,ACCT.JOHN
.
.
.
:SPL
.
.
.

LOCKGLORIN (RINX,TRU,RINPASS);
      (JOB IS SUSPENDED)

      (JOB IS RESUMED)

FWRITE (RES3,...);
.
.
.

UNLOCKGLORIN (RINX);
.
.
.
:EOD
:EOJ
```

Freeing Global RIN's

The owner of a RIN (the user who issued the :GETRIN command to acquire it) can de-allocate the RIN, returning it to the RIN pool managed by MPE/3000. Only the owner can de-allocate a RIN in this fashion. He does this through the :FREERIN command:

```
:FREERIN rin
```

rin The number of the RIN to be de-allocated. (Required parameter.)

INTER-PROCESS LEVEL

The RIN's used at the inter-process level are called *local RIN's*. They are used to exclude simultaneous access of a resource by two or more processes within the same job. Each is a positive integer that is significant only with respect to different processes within that job. Local RIN's are assigned, managed, and released through intrinsic calls.

Acquiring Local RIN's

Just as global RIN's must be acquired by users before they can be used in jobs, local RIN's must be acquired by a job before they can be used by processes within the job. This is done with the following intrinsic call:

PROCEDURE

```
GETLOCRIN (rincount) ;
```

VALUE rincount;

LOGICAL rincount;

OPTION EXTERNAL;

In this intrinsic call, *rincount* is the number of local RIN's to be acquired by the job. The following condition codes can result from the GETLOCRIN call:

- | | |
|-----|---------------------------------------------------------------------------------------------------------------|
| CCE | The request was granted. |
| CCL | Not enough RIN's are available to satisfy this call; none are allocated to the job. |
| CCG | RIN's are already allocated to this job. Additional RIN's cannot be allocated until these RIN's are released. |

Locking and Unlocking Local RIN's

Any local RIN assigned to a job can be locked, by one process at a time, by issuing the LOCKLOCRIN intrinsic call within that process. When this is done, other processes within the job that attempt to lock this RIN are suspended until the locked RIN is released.

PROCEDURE *LOCKLOCRIN (rinnum,lockcond) ;*

VALUE rinnum;

LOGICAL rinnum, lockcond;

OPTION EXTERNAL;

The parameters are

| | |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>rinnum</i> | One of the previously-allocated local RIN's, designated by an integer from 1 to the value specified in the <i>rincount</i> parameter of the GETLOCRIN call. |
| <i>lockcond</i> | A word specifying conditional or unconditional locking: TRUE = Locking will take place unconditionally; if the RIN is not available, the calling process suspends until it is available. (The TRUE condition is passed as a word whose 15th bit is <i>on</i> ; all other bits are ignored.) FALSE = Locking will take place only if the RIN is immediately available. If it is not, control returns to the calling process immediately with the condition code CCG. (The FALSE condition is passed as a word whose 15th bit is <i>off</i> ; all other bits are ignored.) |

The condition codes possible if *lockcond* = TRUE are

| | |
|-----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CCE | The request was granted. If the calling process had already locked the RIN, TRUE is returned to the word <i>lockcond</i> . If the RIN was free, FALSE is returned to <i>lockcond</i> . |
| CCG | (Not returned.) |
| CCL | The request was not granted because the RIN was invalid. |

The condition codes possible if *lockcond* = FALSE are

| | |
|-----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CCE | The request was granted. If the calling process had already locked the RIN, TRUE is returned to the word <i>lockcond</i> . If the RIN was free, FALSE is returned to <i>lockcond</i> . |
| CCG | The request was not granted because the RIN was locked by another process. |
| CCL | The RIN was invalid. |

To unlock this same RIN, the user issues the UNLOCKLOCRIN intrinsic call:

```
PROCEDURE UNLOCKLOCRIN (rinum) ;  
  
VALUE rinum;  
  
LOGICAL rinum;  
  
OPTION EXTERNAL;
```

This call makes the RIN indicated by *rinum* available for locking by other processes in the job. The highest-priority process suspended because this RIN was locked is now activated.

The condition codes possible are

| | |
|-----|--------------------------------------------------------------------------------------|
| CCE | The request was granted. |
| CCG | The request was not granted because the RIN specified is not locked by this process. |
| CCL | The request was not granted because the specified RIN is not allocated to the job. |

To illustrate how the LOCKLOCRIN and UNLOCKLOCRIN intrinsic calls are used, consider two processes (a father process and its son) within a job:

| FATHER PROCESS | SON PROCESS |
|--------------------------|--------------------------|
| ⋮ | ⋮ |
| LP:=FOPEN (LIN, . . .); | LP:=FOPEN (LIN, . . .); |
| ⋮ | ⋮ |
| GETLOCRIN (3); | LOCKLORCIN (1, TRUEVAL); |
| FWRITE (LP, . . .); | FWRITE (LP, . . .); |
| LOCKLORIN (1, TRUEVAL); | ⋮ |
| CREATE (DESCEND, . . .). | ⋮ |
| FWRITE (LP, . . .); | FWRITE (LP, . . .); |
| ⋮ | ⋮ |
| UNLOCKLOCRIN (1); | UNLOCKLOCRIN (1); |
| ⋮ | ⋮ |
| ⋮ | ⋮ |

Suppose that the father process and its son wanted to use RIN (1) to manage a line printer (designated by LP) so that the son process could not use the printer at any time that it was being used by the father process. This could be done as shown in the above coding. When the father process first references LP, the son process is not yet created and the printer need not be locked. However, just prior to creating the son, the father process locks the RIN covering the printer. The father issues all of its print requests before unlocking the printer. Before the son process accesses the printer, it tries to lock it, fails, and is suspended. As soon as the father unlocks the printer, the son process locks it, and issues print requests.

Freeing Local RIN's

The user can free all RIN's currently reserved for his job by issuing this intrinsic call:

PROCEDURE

`FREELOCRIN ;`

OPTION EXTERNAL;

If the GETLOCRIN intrinsic has been called by a process, the FREELOCRIN intrinsic must be called before GETLOCRIN can be called successfully a second time.

The condition codes are

CCE Request granted.

CCG No RIN's are currently reserved for the job.

CCL The request was not granted because at least one RIN is currently locked by a process.

LOCKING AND UNLOCKING FILES

When an FOPEN intrinsic specifying the *dynamic locking aoption* is issued against a disc file, a RIN is established for that file. The user's process can call intrinsics that dynamically lock and unlock the file by alternately acquiring and releasing exclusive use of this RIN. When a file resides on a unit-record device, locking the file (RIN) is equivalent to locking the device itself.

Because the RIN's used in dynamic file locking are global RIN's, the user employing the file-locking intrinsics must follow the rules governing global RIN's. Specific capability-class rules governing file locking are:

1. *Standard Capabilities.* A user's running process (program) can lock only one file at a time.
2. *Process-Handling Optional Capability.* Within the job process structure, only one file can be locked at any one time.
3. *Multiple-RIN Optional Capability.* No restrictions are imposed.

If several processes of the same job are allowed access to a file in an exclusive mode, *local* (as opposed to global) RIN's are available without limitation.

Misuse of the file-locking intrinsics that violate the above rules causes a CCL return from the following intrinsic.

To dynamically lock a file, the user issues the FLOCK intrinsic call:

```
PROCEDURE      FLOCK (filenum,lockcond);
```

```
VALUE filenum, lockcond;
```

```
INTEGER filenum;
```

```
LOGICAL lockcond;
```

```
OPTION EXTERNAL;
```

The parameters are

filenum A word supplying the filenumber of the file to be locked, through SPL/3000 conventions.

lockcond A word specifying conditional or unconditional locking:

TRUE = Locking will take place unconditionally; if the file cannot be locked immediately, the calling process suspends until it can. (Bit 15 = 1.)

FALSE = Locking will take place only if the file's RIN is not currently locked; if the RIN is locked, control returns immediately to the calling process, with condition code CCG.(Bit 15 = 0.)

The condition codes possible if *lockcond* = TRUE are

CCE The request was granted.

CCG (Not returned.)

CCL The request was not granted because this file was not opened with the *dynamic locking aoption*, or the request was to lock more than one file without the *Multiple RIN Optional Capability*.

The condition codes possible if *lockcond* = FALSE are

CCE The request was granted.

CCG The request was not granted because the file was locked by another process.

CCL The request was not granted because this file was not opened with the *dynamic locking aoption*, or the request would lock more than one file without the *Multiple RIN Optional Capability*.

EXAMPLE:

To dynamically lock (unconditionally), the file whose filenumber is stored in the word FILEX the user enters the following call. (TRU is a word whose 15th bit is on.)

FLOCK (FILEX,TRU);

To dynamically unlock a previously-locked file (RIN), the user issues the FUNLOCK intrinsic call:

PROCEDURE

FUNLOCK (filenum);

VALUE filenum;

INTEGER filenum;

OPTION EXTERNAL;

The condition codes possible are

- | | |
|-----|-------------------------------------------------------------------------------------------------------|
| CCE | The request was granted. |
| CCG | The request was not granted because the file had not been locked by the calling process. |
| CCL | The request was not granted because the file was not opened with the <i>dynamic locking aoption</i> . |



SECTION X

MPE/3000 Messages



Users running programs under MPE/3000 at any batch input device or terminal may encounter the following types of error messages:

Command Interpreter Error Messages, reporting fatal errors that occur during the interpretation or execution of an MPE/3000 command.

Command Interpreter Warning Message, reporting unusual conditions that occur during command interpretation or execution but that may not necessarily be detrimental to the processing of the user's job or session.

Segmenter Error Messages, reporting errors that occur while using the MPE/3000 Segmenter.

Run-Time Messages, denoting conditions that abort the user's running program (provided that an appropriate error trap has not been armed).

User Messages, which are messages sent to this user by other users currently running jobs or sessions.

Operator Messages, which are messages sent to this user by the console operator.

System Messages, that denote miscellaneous conditions that terminate or otherwise affect the user's job/session, such as an abort requested by the system supervisor or console operator.

These messages are all discussed in this section.

Other messages may be received only at the MPE/3000 Console. These are:

Console Operator Messages, including:

- Status Messages that indicate the current status of jobs/sessions 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 Failure Messages, including:

- System Halt Messages, reporting the aborting of an uncallable MPE/3000 intrinsic.
- Cold Load Errors, reporting errors encountered while cold-loading the system.

These console operator and system failure messages are discussed in *MPE/3000 Operating System, Console Operator's Guide*.

COMMAND INTERPRETER ERROR MESSAGES

When MPE/3000 detects an error during interpretation or execution of an MPE/3000 command, it suppresses execution of that command and prints an error message on the job/session listing device, in the format shown below.

ERR errnum [,detail] [message]

errnum A number, ranging from 0 to 250, identifying one of the errors described in Figure 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, or by the MPE/3000 Loader
- 250: Error encountered by the MPE/3000 Segmenter.

detail A number that (unless otherwise stated in Figure 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 counted as 1; consider every delimiter, including equal signs and omitted items.

message A message describing the error, also shown in Figure 10-1. In batch jobs, this message appears automatically; in sessions, it may be requested optionally by entering any character other than a carriage return following

ERR errnum [,detail]

When an error occurs while the user is trying to initiate a job or session, the error number is printed but no *message* appears (jobs) or can be requested (sessions).

When an error occurs in a batch job (and no :CONTINUE command precedes the erroneous command, as noted in Section III), all information between the erroneous command and the end-of-job is ignored, and the job is aborted. When an error occurs during an interactive session, control returns to the user, who may then request the *message* display (as described above) or may enter a carriage return to request prompting for a new MPE/3000 command.

| 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/3000 command. Enter a correct command.</p> |
| 2 | <p>ABNORMAL PROGRAM TERMINATION</p> <p>The user's 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 correct command.</p> |
| 4 | <p>INSUFFICIENT CAPABILITY</p> <p>The user does not have the capability required to execute this command or one of its particular options. Check user's 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> |

Figure 10-1. Command Interpreter Error Messages

| ERRNUM | MESSAGE |
|--------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 8 | <p>ILLEGITIMATE ACCESS</p> <p>Access to MPE/3000 was denied because one of the following conditions occurred:</p> <ol style="list-style-type: none"> 1. The user 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, specific group, or home group does not exist. See system manager, supervisor, or operator. 3. A password is required by the account, user, or group (other than home group), and the user 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 user. |
| 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 will be available, or see system supervisor user.</p> |
| 11 | <p>COMMAND NOT YET IMPLEMENTED</p> <p>This command entered is not legal in this version of MPE/3000.</p> |
| 12 | <p>NOT ALLOWED PROGRAMMATICALLY</p> <p>This 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 accommodate more :DATA files. Re-enter command later, or see system supervisor user.</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 user.</p> |
| 15 | <p>SUBSYSTEM NOT FOUND</p> <p>An MPE/3000 subsystem was invoked, but does not exist in the permanent file directory. See system manager user.</p> |

Figure 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> |

Figure 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=outclass</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>outclass</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 was not included for the SYS account. 2. Neither IA nor BA 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>FILESPLACE 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>For an account manager command to create or alter 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>For an account manager command to create or alter 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> |

Figure 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, however, 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 job/session 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 and :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> |

Figure 10-1. Command Interpreter Error Messages (Continued)

| ERRNUM | MESSAGE |
|--------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 64 | <p>UNABLE TO ACCESS TERMINAL FILE</p> <p>The user's job/session 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 not belonging to the user; 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> |

Figure 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 supposedly recorded in the system, is correct. If it is and the user still cannot be referenced, see the system manager user.</p> |
| 80 | <p>SPOOFLE I/O ERROR</p> <p>A disc input/output error occurred or the command was unable to successfully obtain spoo fle disc space. The :STREAM command discontinues processing.</p> |
| 81 | <p>STACK TOO SMALL</p> <p>The :STREAM command cannot obtain sufficient stack 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 Error Codes 20 through 110 in Figure 10-5B. 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> |

Figure 10-1. Command Interpreter Error Messages (Continued)

| ERRNUM | MESSAGE |
|--------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 102 | <p data-bbox="483 302 678 327">NO WRITE RING</p> <p data-bbox="483 367 1357 426">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 data-bbox="483 464 881 489">INCONSISTENT FILE OPERATION</p> <p data-bbox="483 527 1338 585">The command was inconsistent with the access type, record type, or device type for the referenced file. Check for proper consistency.</p> |
| 104 | <p data-bbox="483 621 846 646">PRIVILEGED FILE VIOLATION</p> <p data-bbox="483 684 1175 709">The command attempted unauthorized access to a privileged file.</p> |
| 105 | <p data-bbox="483 747 805 772">INSUFFICIENT DISC SPACE</p> <p data-bbox="483 810 1399 898">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 data-bbox="483 936 794 961">NON-EXISTENT ACCOUNT</p> <p data-bbox="483 999 1284 1058">The command referenced an account not defined in the system. See system manager user.</p> |
| 107 | <p data-bbox="483 1094 761 1119">NON-EXISTENT GROUP</p> <p data-bbox="483 1136 1354 1194">The command referenced a group not defined in the system. See account manager user.</p> |
| 108 | <p data-bbox="483 1230 732 1255">NON-EXISTENT FILE</p> <p data-bbox="483 1283 1321 1341">The command referenced a file not defined in the job temporary or system file domain. Ensure that the file name was entered correctly.</p> |
| 109 | <p data-bbox="483 1377 732 1402">INVALID FILE NAME</p> <p data-bbox="483 1430 1344 1488">The command requested a file with an invalid name. Check the file name spelling and syntax.</p> |
| 110 | <p data-bbox="483 1524 764 1549">DEVICE UNAVAILABLE</p> <p data-bbox="483 1566 1382 1625">The command referenced a device that is not available. Request availability from the operator, or re-specify the device.</p> |
| 111 | <p data-bbox="483 1661 883 1686">INVALID DEVICE SPECIFICATION</p> <p data-bbox="483 1713 1333 1772">The command requested a device with an invalid device specification. Check the specification used, and re-enter command.</p> |
| 112 | <p data-bbox="483 1808 683 1833">NO PASSED FILE</p> <p data-bbox="483 1860 1117 1885">The command required a passed file, but no file was passed.</p> |

Figure 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 process 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/3000 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 job temporary file directory.</p> |
| 117 | <p>DIRECTORY OVERFLOW</p> <p>The command caused the 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 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 creator of the file.</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, which can be interpreted by reference to Message Block D in Figure 10-5D.</p> |

Figure 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 more information about the actual error, and can be interpreted by reference to Message Block C in Figure 10-5C.</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 the user 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 shown in Figure 10-5B (Error Codes 20 through 110). 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 shown in Figure 10-5B (Error Codes 20 through 110). If <i>detail</i> is not displayed, an unexpected end-of-file condition was detected.</p> |

Figure 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 shown in Figure 10-5B (Codes 20 through 110). 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 shown in Figure 10-5B (Codes 20 through 110). 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 shown in Figure 10-5B (Codes 20 through 110). 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> |

Figure 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 groups; 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/3000 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 NORPRIV 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> |

Figure 10-1. Command Interpreter Error Messages (Continued)

COMMAND INTERPRETER WARNING MESSAGES

When certain unusual but generally non-fatal conditions arise during interpretation or execution of an MPE/3000 command, various warning messages may appear on the job/session list device. Most such messages do not terminate batch jobs nor interrupt sessions; a job/session generally continues immediately following the warning message. The message itself appears simply as:

message

The Command Interpreter Warning Messages are summarized in Figure 10-2.

NOTE: In addition to the messages listed in Figure 10-2, other command Interpreter Warning Messages are also output. Figure 10-2 shows only those messages that require explanation; the other messages are self-explanatory.

CAPABILITY EXCEEDS ACCOUNT'S

The user's capability is greater than his account's capability. He is given the maximum capability possible—the intersection of the two sets—for the job/session.

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 the user's 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 Section XI.

EXECUTION PRIORITY REQUESTED EXCEEDS CAPABILITY

The scheduling priority requested at log-on time (through the ;PRI=*priority* parameter of the :HELLO or :JOB command, or its default value (CS)), is greater than the user's maximum priority as defined by the account manager user. The user is given the highest-priority standard subqueue below the BS subqueue that satisfies the user's maximum priority restriction.

INVALID RESPONSE

The user gave an invalid response to a query from MPE/3000 requiring only a YES or NO response; the user is given a second chance to respond.

MAXPRI EXCEEDS ACCOUNT'S

The user's maximum priority value (as defined by the account manager) is greater than that of the account. The user's job/session 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 is a function of the prefix, which in turn is a function of the fully-qualified job name of the sender.) The message is not sent.

PREMATURE JOB TERMINATION

An error during a command in a batch job (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 the user's 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 job to receive a user's message, though defined, is not in EXECUTION state.

Figure 10-2. Command Interpreter Warning Messages

RUN-TIME MESSAGES

A user's program can be aborted as a result of any of the following general types of run-time errors:

- Special violations—those detected through the internal interrupt structure (such as arithmetic trap errors, parity errors, bounds violations, and so forth) and other violations detected by MPE/3000 (such as stack overflows or invalid stack markers). These are PROGRAM ERRORS and are described in figure 10-3.
- Explicit calls to the QUIT intrinsic (Section VIII).
- Explicit calls to the QUITROG intrinsic (Section VIII).
- Violations of other callable intrinsics, such as passing of illegal parameters or invoking of an intrinsic without having the required capability class or a valid register environment. (These errors are listed in figure 10-5; while, the intrinsics are listed in figure 10-4 and the errors encountered by them are listed in figures 10-6 through 10-12.)

If an appropriate error trap has been armed, control transfers to the trap procedure which may attempt recovery or take some other action. But if no trap has been armed for the type of error encountered, MPE/3000 terminates the user's process and transmits a *run-time (abort) message* to the user's output device. In a multi-process structure, QUIT aborts only the violating process but all other errors abort the entire program.

If the aborted program was running in a batch job, the job is removed from the system (if no :CONTINUE command overrides termination).

If it was running in a session, control of the session is returned to the user at the terminal.

NOTE: An abort-error will occur if a user process invokes certain callable intrinsics when the DB register is not pointing to its normal position (e.g. DB is pointing at an extra data segment). If this happens and a user trap procedure is invoked, the DB register is reset to the normal position before the trap procedure is entered.

The format for a run-time (abort) error message is:

ABORT:pname.segment.location:sname.segment.location:message

$\underbrace{\hspace{10em}}_{p\text{-field}} \quad \underbrace{\hspace{10em}}_{s\text{-field}} \quad \underbrace{\hspace{5em}}_{m\text{-field}}$

msgtype # {msgno} : message [.PARM {#} number]

The *p-field* indicates the last location of the last instruction executed in the user program prior to the abort.

The *s-field* is output only if the abort occurred when executing code belonging to a library segment, referenced by the user program. The field provides the instruction location within the library segment that initiated the abort.

The *m-field* contains the error message text, as described below.

Within each field, the parameters are:

pname The name of the program file containing the user's program, and optionally, the group and account name.

In the special case of a process having been PROCREATED from a segment in a segmented library (SL) (for example, the Command Interpreter), an asterisk (*) is output followed by the SL name in symbolic form (sname, below).

sname The symbolic name of the SL in which the segment exists

 SYSL - System SL

 PUSL - Public SL

 GRSL - Group SL

segment The logical number of the code segment with respect to either the program or SL, whichever is appropriate.

location The location in the code segment. This is expressed in terms of the displacement (P-PB), where P is the absolute address of the instruction and PB the absolute address of the base of the code segment.

NOTE: *Octal numbers are indicated by a percent sign (%) preceding the number.*

If the stack is completely destroyed and no valid stack markers can be found that define a user environment, then the above-defined subfields will be output containing a question mark (?).

msgtype Any of the error messages defined in figures 10-3 through 10-12 and corresponds to the names of the following tables.

msgno A message number, which is an index into the *msgtype* table.

message The text of the message which can be found along with the message number in the message type table.

number The number of the invalid parameter passed to an intrinsic (the message will read: PARAM # *number*) or is the parameter passed to the QUIT or QUITPROG intrinsic (the message will read: PARAM =).

EXAMPLES

*The following example shows that
BINARY was called with an invalid byte address.*

```
ABORT:BIN.ED.MPE.%0.%12  
ERROR:INTRINSIC #62:BINARY  
RUN-TIME ERROR #5:PARAMETER ADDRESS VIOLATION.PARAM #1
```

The following program was in an infinite loop doing a DUP instruction:

```
ABORT:OV.ED.MPE.%0.%177777  
PROGRAM ERROR #20:STACK OVERFLOW
```

A return was made from a non-privileged segment to a privileged segment in the following example:

```
ABORT:PRIV.ED.MPE.%0.%3  
PROGRAM ERROR #6:PRIVILEGED INSTRUCTION
```

The following example shows the program called QUIT Intrinsic with a parameter of 15:

```
ABORT:QUIT.ED.MPE.%0.%1  
PROGRAM ERROR #18;PROCESS QUIT.PARAM=15
```

The following program was in an infinite loop doing a DEL instruction:

```
ABORT:UF.ED.MPE.%0.%1  
PROGRAM ERROR #29:STACK UNDERFLOW
```

The following example shows that nearly all CST entries were ALLOCATED and the program tried to create a process which required more CST's than were available:

```
ABORT:EDITOR.PUB.SYS.%2.%7  
ERROR:INTRINSIC #100:CREATE  
CREATE ERROR #30:LOAD ERROR  
LOADER ERROR #65:UNABLE TO OBTAIN CST ENTRIES
```

The following program tried to activate a non-existent process:

```
ABORT:EDITOR.PUB.SYS.%2.%13  
ERROR:INTRINSIC #104:ACTIVATE  
ACTIVATE ERROR #21:ACTIVATION OF MAIN PROCESS NOT ALLOWED
```

| MSGNO | MESSAGE | COMMENT |
|-------|-----------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1 | INTEGER OVERFLOW | } Logic error in the program. |
| 2 | FLOATING POINT OVERFLOW | |
| 3 | FLOATING POINT UNDERFLOW | |
| 4 | INTEGER DIVIDE BY ZERO | |
| 5 | FLOATING POINT DIVIDE BY ZERO | |
| 6 | PRIVILEGED INSTRUCTION | |
| 7 | ILLEGAL INSTRUCTION | |
| 8 | EXTENDED PRECISION OVERFLOW | |
| 9 | EXTENDED PRECISION UNDERFLOW | |
| 10 | EXTENDED PRECISION DIVIDE BY ZERO | |
| 11 | DECIMAL OVERFLOW | |
| 12 | INVALID ASCII DIGIT | |
| 13 | INVALID DECIMAL DIGIT | |
| 14 | INVALID WORD COUNT | |
| 15 | INVALID DECIMAL OPERAND LENGTH | |
| 16 | DECIMAL DIVIDE BY ZERO | |
| 17 | STT UNCALLABLE | |
| 18 | PROCESS QUIT.PARAM=<number> | } <number> is the value passed to the QUITPROG or QUIT intrinsic (Section VIII) by the terminating process. (This value is output only if it is <i>not</i> zero). |
| 19 | PROGRAM QUIT.PARAM=<number> | |
| 20 | STACK OVERFLOW | } Logic error in the program. Probably looping and adding to stack. May require larger MAX-DATA when preparing program. |
| 21 | PROGRAM KILLED | } Program aborted from an external source. |
| 22 | INVALID STACK MARKER | } Possible hardware problem. <i>Please See communication #2</i> |
| 23 | ADDRESS VIOLATION | |
| 24 | BOUNDS VIOLATION | |
| 25 | NON-RESPONDING MODULE | |
| 26 | DATA PARITY | |
| 27 | MEMORY PARITY | |
| 28 | SYSTEM PARITY | |
| 29 | STACK UNDERFLOW | } Logic error in program. Probably looping and popping stack. |
| 30 | CST VIOLATION | } Invalid CST or STT discovered by hardware. Explicit PCAL from TOS may have referenced non-existent CST or STT. May be bad program file. |
| 31 | STT VIOLATION | |

Figure 10-3. Program Errors

| MSGNO (INTRINSIC NO.) | MESSAGE (NAME) | MSGNO (INTRINSIC NO.) | MESSAGE (NAME) |
|--------------------------|-------------------|--------------------------|-------------------|
| 1 | FOPEN | 65 | PRINT |
| 2 | FREAD | 66 | PRINTOP |
| 3 | FWRITE | 67 | PRINTOREPLY |
| 4 | FUPDATE | 68 | COMMAND |
| 5 | FSPACE | 69 | WHO |
| 6 | FPOINT | 70 | SEARCH |
| 7 | FREADDIR | 71 | MYCOMMAND |
| 8 | FCLOSE | 72 | SETJCW |
| 10 | FCHECK | 73 | GETJCW |
| 11 | FGETINFO | 74 | DBINARY |
| 12 | FREADSEEK | 75 | DASCII |
| 13 | FCONTROL | 76 | QUIT |
| 14 | FSETMODE | 77 | STACKDUMP |
| 15 | FLOCK | 78 | SETDUMP |
| 16 | FUNLOCK | 79 | RESETDUMP |
| 17 | FRENAME | 80 | LOADPROC |
| 18 | FRELATE | 81 | UNLOADPROC |
| 19 | FREADLABEL | 82 | INITUSLF |
| 20 | FWRITELABEL | 83 | ADJUSTSLF |
| 21 | PRINTFILEINFO | 84 | EXPANDUSLF |
| 22 | IOWAIT | 99 | DEBUG |
| 30 | GETLOCRIN | 100 | CREATE |
| 31 | FRELOCRIN | 102 | KILL |
| 32 | LOCKLOCRIN | 103 | SUSPEND |
| 33 | UNLOCKLOCRIN | 104 | ACTIVATE |
| 34 | LOCKGLORIN | 105 | GETORIGIN |
| 35 | UNLOCKGLORIN | 106 | MAIL |
| 40 | TIMER | 107 | SENDMAIL |
| 42 | PROCTIME | 108 | RECEIVEMAIL |
| 43 | CALENDAR | 109 | FATHER |
| 44 | CLOCK | 110 | GETPROCINFO |
| 45 | PAUSE | 112 | GETPROCID |
| 50 | XARITRAP | 120 | GETPRIORITY |
| 51 | ARITRAP | 130 | GETDSEG |
| 52 | XLIBTRAP | 131 | FREEDSEG |
| 53 | XSYSTRAP | 132 | DMOVIN |
| 54 | XCONTRAP | 133 | DMOVEOUT |
| 55 | RESETCONTROL | 134 | ALTDSEG |
| 56 | CAUSEBREAK | 135 | DLSIZE |
| 60 | TERMINATE | 136 | ZSIZE |
| 62 | BINARY | 139 | SWITCHDB |
| 63 | ASCII | 191 | PTAPE |
| 64 | READ,READX | 200 | GETPRIVMODE |
| | | 201 | GETUSERMODE |

Figure 10-4. Intrinsic Numbers vs. Intrinsics

| | MSGNO | MESSAGE |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|---------------------------------|
| Run-time errors are discovered by MPE performing parameter checking before attempting certain operations. These errors are caused by a logic error in the program. | 1 | ILLEGAL DB REGISTER |
| | 2 | ILLEGAL CAPABILITY |
| | 3 | OMITTED PARAMETER |
| | 4 | INCORRECT S REGISTER |
| | 5 | PARAMETER ADDRESS VIOLATION |
| | 6 | PARAMETER AND ADDRESS VIOLATION |
| | 7 | ILLEGAL PARAMETER |
| | 8 | PARAMETER VALUE INVALID |
| | 9 | INCORRECT Q REGISTER |

Figure 10-5. Run-Time Errors

| MSGNO | MESSAGE |
|-------|-----------------------------------------------|
| 0 | END OF FILE |
| 20 | INVALID OPERATION |
| 21 | DATA PARITY ERROR |
| 22 | SOFTWARE TIME-OUT |
| 23 | END OF TAPE |
| 24 | UNIT NOT READY |
| 25 | NO WRITE-RING ON TAPE |
| 26 | TRANSMISSION ERROR |
| 27 | I/O TIME-OUT |
| 28 | TIMING ERROR OR DATA OVERRUN |
| 29 | SIO FAILURE |
| 30 | UNIT FAILURE |
| 31 | END OF LINE |
| 32 | SOFTWARE ABORT |
| 33 | DATA LOST |
| 34 | UNIT NOT ON-LINE |
| 35 | DATA-SET NOT READY |
| 36 | INVALID DISC ADDRESS |
| 37 | INVALID MEMORY ADDRESS |
| 38 | TAPE PARITY ERROR |
| 39 | RECOVERED TAPE ERROR |
| 40 | OPERATION INCONSISTENT WITH ACCESS TYPE |
| 41 | OPERATION INCONSISTENT WITH RECORD TYPE |
| 42 | OPERATION INCONSISTENT WITH DEVICE TYPE |
| 43 | WRITE EXCEEDS RECORD SIZE |
| 44 | UPDATE AT RECORD ZERO |
| 45 | PRIVILEGED FILE VIOLATION |
| 46 | OUT OF DISC SPACE |
| 47 | I/O ERROR ON FILE LABEL |
| 48 | INVALID OPERATION DUE TO MULTIPLE FILE ACCESS |
| 49 | UNIMPLEMENTED FUNCTION |
| 50 | NONEXISTENT ACCOUNT |

Figure 10-6. File System Errors

| MSGNO | MESSAGE |
|-------|------------------------------------------------|
| 51 | NONEXISTENT GROUP |
| 52 | NONEXISTENT PERMANENT FILE |
| 53 | NONEXISTENT TEMPORARY FILE |
| 54 | INVALID FILE REFERENCE |
| 55 | DEVICE UNAVAILABLE |
| 56 | INVALID DEVICE SPECIFICATION |
| 57 | OUT OF VIRTUAL MEMORY |
| 58 | NO PASSED FILE |
| 59 | STANDARD LABEL VIOLATION |
| 60 | GLOBAL RIN UNAVAILABLE |
| 61 | OUT OF GROUP DISC SPACE |
| 62 | OUT OF ACCOUNT DISC SPACE |
| 63 | USER LACKS NON-SHARABLE DEVICE CAPABILITY |
| 64 | USER LACKS MULTI-RUN CAPABILITY |
| 65 | PUNCH HOPPER EMPTY |
| 66 | PLOTTER LIMIT SWITCH REACHED |
| 67 | PAPER TAPE ERROR |
| 68 | INSUFFICIENT SYSTEM RESOURCES |
| 69 | I/O ERROR |
| 71 | TOO MANY FILES OPEN |
| 72 | INVALID FILE NUMBER |
| 73 | BOUNDS VIOLATION |
| 80 | SPOOFLE SIZE EXCEEDS CONFIGURATION |
| 81 | NO "SPOOL" CLASS IN SYSTEM |
| 82 | INSUFFICIENT SPACE FOR SPOOFLE |
| 83 | I/O ERROR ON SPOOFLE |
| 84 | DEVICE UNAVAILABLE FOR SPOOFLE |
| 85 | OPERATION INCONSISTENT WITH SPOOLING |
| 86 | NONEXISTENT SPOOFLE |
| 87 | BAD SPOOFLE BLOCK |
| 89 | POWER FAILURE |
| 90 | EXCLUSIVE VIOLATION: FILE BEING ACCESSED |
| 91 | EXCLUSIVE VIOLATION: FILE ACCESSED EXCLUSIVELY |
| 92 | LOCKWORD VIOLATION |
| 93 | SECURITY VIOLATION |
| 94 | USER IS NOT CREATOR |
| 95 | READ COMPLETED DUE TO BREAK |
| 96 | DISC I/O ERROR |
| 97 | NO CONTROL Y PIN |
| 98 | READ TIME OVERFLOW |
| 99 | BOT AND BACKSPACE TAPE |
| 100 | DUPLICATE PERMANENT FILE NAME |
| 101 | DUPLICATE TEMPORARY FILE NAME |
| 102 | I/O ERROR ON DIRECTORY |
| 103 | PERMANENT FILE DIRECTORY OVERFLOW |
| 104 | TEMPORARY FILE DIRECTORY OVERFLOW |
| 105 | BAD VARIABLE BLOCK STRUCTURE |
| 106 | EXTENT SIZE EXCEEDS MAXIMUM |
| 107 | INSUFFICIENT SPACE FOR USER LABELS |
| 108 | DEFECTIVE FILE LABEL ON DISC |
| 109 | INVALID VARIABLE CONTROL |
| 110 | ATTEMPT TO SAVE PERMANENT FILE AS TEMPORARY |
| 111 | USER LACKS SF CAPABILITY |

Figure 10-6. File System Errors (Continued)

| MSGNO | MESSAGE | COMMENT |
|-------|------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| 20 | ILLEGAL LIBRARY SEARCH | |
| 21 | UNKNOWN ENTRY POINT | |
| 22 | TRACE SUBSYSTEM NOT PRESENT | |
| 23 | STACK SIZE TOO SMALL | |
| 24 | MAXDATA TOO LARGE | MAXDATA must be no greater than 31,232 |
| 25 | DATA SEGMENT TOO LARGE | |
| 26 | PROGRAM LOADED IN OPPOSITE MODE | A privileged program is currently loaded in the opposite PRIV/NON-PRIV mode. |
| 27 | SL BINDING ERROR | |
| 28 | INVALID SYSTEM SL FILE | |
| 29 | INVALID PUBLIC SL FILE | |
| 30 | INVALID GROUP SL FILE | |
| 31 | INVALID PROGRAM FILE | |
| 32 | INVALID LIST FILE | |
| 33 | CODE SEGMENT TOO LARGE | System may be reconfigured by system supervisor/manager for larger code segment. |
| 34 | PROGRAM USES MORE THAN ONE EXTENT | Programs must be located in contiguous disc space. Build new program file with larger extent size. |
| 35 | DATA SEGMENT TOO LARGE | Data segment greater than 32,767 words, the hardware limitation. |
| 36 | DATA SEGMENT TOO LARGE | System may be reconfigured by system supervisor/manager for larger data segment. |
| 37 | TOO MANY CODE SEGMENTS | A program file can contain a maximum of 152 segments. |
| 38 | TOO MANY CODE SEGMENTS | System may be reconfigured by system supervisor/manager for more code segments. |
| 39 | ILLEGAL CAPABILITY | |
| 40 | TOO MANY PROCEDURES LOADED | |
| 41 | UNKNOWN PROCEDURE NAME | |
| 42 | INVALID PROCEDURE NUMBER | |
| 43 | ILLEGAL PROCEDURE UNLOAD | |
| 50 | UNABLE TO OPEN SYSTEMS SL FILE | |
| 51 | UNABLE TO OPEN PUBLIC SL FILE | |
| 52 | UNABLE TO OPEN GROUP SL FILE | |
| 53 | UNABLE TO OPEN PROGRAM FILE | |
| 54 | UNABLE TO OPEN LIST FILE | |
| 55 | UNABLE TO CLOSE SYSTEM SL FILE | |
| 56 | UNABLE TO CLOSE PUBLIC SL FILE | |
| 57 | UNABLE TO CLOSE GROUP SL FILE | |
| 58 | UNABLE TO CLOSE PROGRAM FILE | |
| 59 | UNABLE TO CLOSE LIST FILE | |
| 60 | EOF OF I/O ERROR ON SYSTEM SL FILE | |
| 61 | EOF OR I/O ERROR ON PUBLIC SL FILE | |
| 62 | EOF OR I/O ERROR ON GROUP SL FILE | |
| 63 | EOF OR I/O ERROR ON PROGRAM FILE | |
| 64 | EOF OR I/O ERROR ON LIST FILE | |
| 65 | UNABLE TO OBTAIN CST ENTRIES | System is loaded to capacity. Either a running program must terminate, or an ALLOCATED program or procedure not in use must be DEALLOCATED. |

Figure 10-7. Loader Errors

| MSGNO | MESSAGE | COMMENT |
|-------|----------------------------------------|-----------------------------------------------------------------------------------|
| 66 | UNABLE TO OBTAIN PROCESS DST ENTRY | System is loaded and there are insufficient resources to create the load process. |
| 67 | UNABLE TO OBTAIN MAIL DATA SEGMENT | |
| 68 | UNABLE TO CREATE LOAD PROCESS | |
| 70 | SEGMENT TABLE OVERFLOW | |
| 71 | UNABLE TO OBTAIN SUFFICIENT DL STORAGE | |
| 72 | ATTIO ERROR | |
| 73 | UNABLE TO OBTAIN VIRTUAL MEMORY | |
| 74 | DIRECTORY I/O ERROR | |
| 75 | PRINT I/O ERROR | |
| 76 | ILLEGAL DLSIZE | |
| 80 | PROGRAM ALREADY ALLOCATED | |
| 81 | ILLEGAL PROGRAM ALLOCATION | |
| 82 | PROGRAM NOT ALLOCATED | |
| 83 | ILLEGAL PROGRAM DEALLOCATION | |
| 84 | PROCEDURE ALREADY ALLOCATED | |
| 85 | ILLEGAL PROCEDURE ALLOCATION | |
| 86 | PROCEDURE NOT ALLOCATED | |
| 87 | ILLEGAL PROCEDURE DEALLOCATION | |

Figure 10-7. Loader Errors (Continued)

| MSGNO | MESSAGE | COMMENT |
|-------|-------------------------------------------------------|-------------------------------------------------------------------------------------------------|
| 20 | UNKNOWN SUBQUEUE NAME | Error occurred in loader. System is loaded and there are insufficient PCB's to load process. |
| 21 | SUBQUEUE 'A' REQUESTED WITHOUT FROZEN STACK | |
| 23 | INSUFFICIENT CAPABILITY FOR NON-STANDARD SUBQUEUE | |
| 24 | UNKNOWN PORTION OF MASTER QUEUE | |
| 25 | INSUFFICIENT CAPABILITY FOR MASTER QUEUE | |
| 26 | ABSOLUTE PRIORITY REQUESTED WITHOUT CAPABILITY | |
| 27 | ILLEGAL PRIORITY CLASS SPECIFIED | |
| 28 | PRIORITY OMITTED WHILE FATHER PROCESS IN MASTER QUEUE | |
| 29 | PRIORITY RANK RESERVED TO SUPERVISOR CAPABILITY | |
| 30 | LOAD ERROR | |
| 31 | LACK OF SYSTEM RESOURCE | |
| 32 | MAXIMUM ACCOUNT PRIORITY EXCEEDED | |

Figure 10-8. CREATE Errors

| MSGNO | MESSAGE | COMMENT |
|----------|------------------------------------------------------------------------------------|------------------------|
| 20 21 | ACTIVATION OF SYSTEM PROCESS NOT ALLOWED ACTIVATION OF MAIN PROCESS NOT ALLOWED | Process may not exist. |

Figure 10-9. ACTIVATE Errors

| MSGNO | ERROR |
|-------|-------------------------|
| 20 | INSUFFICIENT CAPABILITY |

Figure 10-10. SUSPEND Error

| MSGNO | ERROR |
|-------|----------------------------------------------|
| 20 | PARSED PARAM OF COMIMAGE > 255 CHARACTERS |

Figure 10-11. MYCOMMAND Error

| MSGNO | ERROR |
|-------|------------------------------------|
| 20 | INCORRECT PASSWORD FOR RIN |
| 21 | ONLY ONE RIN CAN BE LOCKED |
| 22 | RIN IS NOT ALLOCATED |
| 23 | FIN IS TOO LARGE FOR THE RIN TABLE |
| 24 | RIN IS NOT GLOBAL RIN |

Figure 10-12. LOCKGLORIN Errors

USER MESSAGES

When a user's batch job or session receives a message from another user's job or session, that message appears in the following format:

FROM/[jsname,]username.acctname/message

| | | |
|-----------------|---|---------------------------------------------------------------------------------------------------------------|
| <i>jsname</i> | } | The names of the transmitting job/session and user, and the name of the account under which they are running. |
| <i>username</i> | | |
| <i>acctname</i> | | |
| <i>message</i> | | The message. |

EXAMPLE

A user identified as BOB running a session named S, under an account named A, sends a message to a user telling him that he is changing the name of a file frequently used by both persons; the receiving user would see the following message:

FROM/S,BOB.A/FILE NAMED BAKER IS NOW RENAMED JONES.

OPERATOR MESSAGES

When a user's batch job or session receives a message from the console operator, that message appears in one of two formats, depending on its degree of urgency. Urgent messages which pre-empt any form of input/output being conducted on the standard list device, appear in this format:

WARN/message

message is the message text.

Less serious messages used for normal communication between the operator and user do not pre-empt input/output in progress, and appear on the standard list device in this format:

FROM/OPERATOR/message

message is the message text.

SYSTEM MESSAGES

Miscellaneous conditions that terminate or otherwise affect a users' job/session are reported through system messages, shown in Figure 10-6. These messages may appear, asynchronously, during the course of a running job/session on the standard list device.

CAN NOT INITIATE NEW SESSIONS NOW

New sessions cannot be initiated due to one of the following problems:

1. Insufficient system resources to start job.
2. Session limit would be exceeded (see = LIMIT and = LOGOFF).
3. Requestor's input priority (INPRI =) is not greater than current <jobfence> (see = JOBFENCE).

NOTE: System managers and system supervisors can bypass rejections due to 2 and 3, above, by supplying HIPRI on :HELLO command.

* {
SESSION
JOB } ABORTED BY SYSTEM MANAGEMENT.*

The job/session has been aborted by the computer operator or system supervisor user through the appropriate command. An immediate log-off then takes place.

* {
SESSION
JOB } HAS EXCEEDED TIME LIMIT.*

The job/session has exceeded the time limit which was specified in the TIME=parameter of the JOB/HELLO 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.

Figure 10-13. System Messages

FILE INFORMATION DISPLAY

In addition to Command Interpreter and run-time (abort) error messages, certain file input/output errors result in the output of a file information display. For files not yet opened, or for which the FOPEN intrinsic failed, this display appears as in the example below.

```

+-F-I-L-E---I-N-F-O-R-M-A-T-I-O-N---D-I-S-P-L-A-Y+
①→! FILE NUMBER #           IS UNDEFINED.           !
②→! ERROR NUMBER: 56        RESIDUE: 0              !
③→! BLOCK NUMBER: 0         NUMREC: 0               !
+-----+

```

In this display, the lines indicated show the following information:



| Line | Content |
|------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1 | A message stating why the file could not be opened. |
| 2 | The appropriate <i>error number</i> relating to Figure 10-5B; in this case, an invalid device specification occurred. The <i>residue</i> , which is the number of bytes <i>not</i> transferred in an input/output request; since no such request applies in this case, this is <i>zero</i> . |
| 3 | The <i>block number</i> , indicating the physical record where the error was encountered. The <i>numrec</i> (number of logical records in the block). In this case, since the file was not opened, both <i>block number</i> and <i>numrec</i> are zero. |

For files that were open when a CCG (end-of-file error) or CCL (irrecoverable file error) condition code was returned, the file information display appears as shown in this example:

```

+-F-I-L-E---I-N-F-O-R-M-A-T-I-O-N---D-I-S-P-L-A-Y+
①→! FILE NAME IS FTN05                               !
②→! FOPTIONS: SYS,A, $STDIN,U,N.FEQ                 !
③→! AOPTIONS: INPUT,SREC,NOLOCK,DEF,NOBUFF          !
④→! DEVICE TYPE: 16      DEVICE SUBTYPE: 0          !
⑤→! LDEV: 11      DRT: 13      UNIT: 0              !
⑥→! RECORD SIZE: 72     BLOCK SIZE: 72      (BYTES) !
⑦→! EXTENT SIZE: 0      MAX EXTENTS: 0           !
⑧→! RECPTR: 0          RECLIMIT: 0              !
⑨→! LOGCOUNT: 0       PHYSCOUNT: 0             !
⑩→! EOF AT: 0          LABEL ADDR: %01300000000    !
⑪→! FILE CODE: 0      ID IS          ULABELS: 0    !
⑫→! PHYSICAL STATUS: 0000101100000000            !
⑬→! ERROR NUMBER: 0    RESIDUE: 0              !
14 ! BLOCK NUMBER: 0    NUMREC: 1              !
+-----+

```


| Line | Contents |
|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 3 (Cont.) | Multi-record SREC = Single record access (as in this case). Option: MREC = Multi-record access. |
| | Dynamic NOLOCK = No locking permitted (as in this case). Locking LOCK = Locking permitted. Option: |
| | Exclusive DEF = Default specification (as in this case). Access EXC = Exclusive access allowed. Option: SEA = Semi-exclusive access allowed. SHR = Sharable file. |
| | Buffering: BUFFER = Automatic buffering. NOBUFF = Inhibit buffering (as in this case). |
| 4,5 | The <i>Device Type</i> , <i>Device Subtype</i> , <i>LDEV (Logical Device Number)</i> , <i>DRT (Device Reference Table Entry Number)</i> and <i>Unit</i> of the device on which the file resides. (These are 16, 0, 11, 18, and 0 respectively, in this case.) |
| 6 | The <i>record size</i> and <i>block size</i> of the offending record, in <i>bytes</i> . (In this case, these are both specified as 72 bytes.) |
| 7 | The <i>extent size</i> (of the current extent) and the <i>max extents</i> (maximum number of extents) allowed the file. |
| 8 | The <i>recptr</i> (current record pointer) and <i>reclimit</i> (limit on number of records in the file). |
| 9 | The <i>logcount</i> (present count of logical records) and <i>physcount</i> (present count of physical records) in the file). |
| 10 | The <i>EOF at</i> (location of the current end-of-file) and the <i>label addr</i> (location of the header label of the file). |
| 11 | The <i>file code</i> , <i>id</i> (name of creating user), and <i>ulabels</i> (number of user-created labels) for the file. |
| 12 | The <i>physical status</i> of the file. |
| 13 | The <i>error number</i> and <i>residue</i> , as described under the abbreviated file information display format, above. |
| 14 | The <i>block number</i> and <i>numrec</i> , as described under the abbreviated file information display format, above. |

SEGMENTER ERROR MESSAGES

Segmenter process errors are of the following format:

```

{ *** ERROR ***
  *** FILE ERROR
  *** WARNING *** } [numericparam] [stringparam]
  
```

ERROR msgnum message

Following is an example of a message which contains both a *numericparam* (40) and a *stringparam* (AUX. USL FILE). The message number is 84.

```

*** FILE ERROR 40 AUX. USL FILE
ERROR #84 UNEXPECTED I/O ERROR
  
```

Figure 10-7 lists the message portion of all Segmenter process errors along with a brief explanation of the message.

| Error No. | Error Message | Comments |
|-----------|------------------------------------------|-------------------------------------------------------------------------------------------------------------------|
| 0 | ILLEGAL ENTRY | Bad USL generated by compiler. <i>numericparam</i> is decimal address of entry in USL. |
| 1 | ILLEGAL HEADER | Bad USL generated by compiler. |
| 2 | ATTEMPT TO EXCEED MAXIMUM DIRECTORY SIZE | USL file maximum directory size is %77777. |
| 3 | AVAILABLE DIRECTORY SPACE EXHAUSTED | USL file too small. Compiler output may be in error. |
| 4 | AVAILABLE INFO SPACE EXHAUSTED | USL file too small. Compiler output may be in error. |
| 5 | USL FILE NOT DESIGNATED | Tried to -USE, -PREPARE, -NEWSEG, -LISTUSL, -COPY, -ADDRL, -ADDSL, -HIDE, -REVEAL or -PURGEFRBM and USL not open. |
| 6 | ILLEGAL USL FILE SPECIFICATION | USL file length is less than 5 records or greater than 32727 records. |
| 7 | UNABLE TO OPEN USL FILE | FOPEN fail in -AUXUSL, -BUILDUSL or -USL. <i>numericparam</i> is FCHECK value. |
| 8 | INVALID USL FILE | File code is not USL or USL file is bad. |
| 9 | UNABLE TO CLOSE USL FILE | FCLOSE failure during -EXIT. <i>numericparam</i> is FCHECK value. |
| 10 | UNABLE TO CLOSE SL FILE | FCLOSE fail during -EXIT or on previous SL during -SL. <i>numericparam</i> is FCHECK value. |
| 11 | AVAILABLE FILE SPACE EXHAUSTED | RL or SL file is full. |

Figure 10-7. Segmenter Error Messages

| Error No. | Error Message | Comments |
|-----------|------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 12 | ENTRY POINT ALREADY DEFINED | <i>stringparam</i> is entry name. |
| 13 | SEGMENT CONTAINS PROGRAM UNITS OTHER THAN PROCEDURES | Outer block is present. For SPL, recompile in subprogram mode. |
| 14 | SEGMENT REQUIRES GLOBAL STORAGE | FORTRAN GLOBAL, or SPL GLOBAL or OWN variables were specified. <i>stringparam</i> is entry name. |
| 15 | SEGMENT ALREADY DEFINED | -ADDSL segname and segname already exists in segment. <i>numericparam</i> is <i>segname</i> . |
| 16 | SL FILE NOT DESIGNATED | Tried to -PURGESL, -ADDSL or -LISTSL and SL not open. |
| 17 | ILLEGAL SL FILE SPECIFICATION | SL file length is less than 4 records or greater than %77777 records. |
| 18 | UNABLE TO OPEN SL FILE | FOPEN failure during -BUILDSL or -SL. <i>numericparam</i> is FCHECK value. |
| 19 | INVALID SL FILE | |
| 20 | ILLEGAL RL FILE SPECIFICATION | RL file length is less than 4 records or greater than %77777 records. |
| 21 | RL FILE NOT DESIGNATED | Tried to -ADDRL, -LISTRL or -PURGERL without opening RL. |
| 22 | INVALID RL FILE | File code is not RL or RL file is bad. |
| 23 | UNABLE TO CLOSE RL FILE | FCLOSE Failure. <i>numericparam</i> is FCHECK value. |
| 28 | PROCEDURE HAS NOT USABLE ENTRY POINT | |
| 30 | UNABLE TO OPEN RL FILE | FOPEN Failure. <i>numericparam</i> is FCHECK value. |
| 32 | INVALID PROGRAM FILE | File code is not program. |
| 33 | ILLEGAL CAPABILITY SPECIFICATION | Program capability specification is greater than user's. |
| 34 | MORE THAN ONE EXTENT USED | Not all code is in one extent. Program will not run unless code is contiguous on disc (the loader will detect this problem). Build a new program file with larger extents. This is a WARNING message. |
| 35 | NO PROGRAM TO PREPARE | No program in USL. |
| 36 | UNABLE TO CLOSE PROGRAM FILE | FCLOSE failure. <i>numericparam</i> is FCHECK value. |
| 37 | UNABLE TO OPEN PROGRAM FILE | FOPEN failure. <i>numericparam</i> is FCHECK value. |
| 38 | DATA SEGMENT OVERFLOW | Data is greater than %37777. |

Figure 10-7. Segmenter Error Messages (Continued)

| Error No. | Error Message | Comments |
|-----------|-------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 39 | TOO MANY CODE SEGMENTS | More than 255 segments in an SL, or more than 152 segments in a program file. (The structure of a program file is limited to 152 segments.) |
| 40 | CODE SEGMENT OVERFLOW | Code segment greater than %37774. |
| 41 | STT OVERFLOW | Too many PCAL instructions. <i>stringparam</i> is entry name where overflow occurred. |
| 42 | SEGMENT HAS NO USABLE ENTRY POINT | |
| 43 | UNABLE TO ACCESS PROCEDURE | Illegal P-Label or params not matching. |
| 44 | REQUIRES PRIVILEGED MODE CAPABILITY | User needs privileged mode capability to add privileged segment. |
| 45 | ACTUAL PARAMETERS INCOMPATIBLE WITH FORMAL PARAMETERS | (FORTRAN Program) <i>stringparam</i> is procedure name. |
| 46 | PROGRAM UNIT CONTAINS FATAL ERROR | <i>stringparam</i> is unit name. |
| 47 | PROGRAM UNIT CONTAINS NON-FATAL ERROR | <i>stringparam</i> is unit name. This is a WARNING message. |
| 48 | CODE SEGMENT MAY BE TOO LARGE | This is a WARNING message. |
| 60 | NO OUTER BLOCK IS ACTIVE | |
| 61 | MORE THAN ONE OUTER BLOCK IS ACTIVE | |
| 62 | MORE THAN ONE OUTER BLOCK HAS ACTIVE ENTRY POINTS | |
| 63 | EXTERNAL VARIABLE NOT DECLARED GLOBAL | <i>stringparam</i> is entry name. |
| 64 | EXTERNAL VARIABLE INCOMPATIBLE WITH GLOBAL VARIABLE | <i>stringparam</i> is entry name. |
| 66 | TOO MANY COMMON DATA LABELS | (BASICOMP or FORTRAN program) The number of COMMON data labels has caused the primary DB area to exceed 255 (decimal). Reduce the number of COMMON data labels. |
| 67 | COMMON DECLARED WITH DIFFERENT SIZE | <i>stringparam</i> is common name. |
| 68 | ATTEMPT TO USE BLOCK DATA ON NON-EXISTENT COMMON | (FORTRAN program) |
| 69 | ATTEMPT TO USE BLOCK DATA ON INCOMPATIBLE COMMON | (FORTRAN program) |

Figure 10-7. Segmenter Error Messages (Continued)

| Error No. | Error Message | Comments |
|-----------|-----------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 70 | ILLEGAL STACK SIZE | Stack size less than 0. |
| 71 | ILLEGAL DL SIZE | DL size less than 0. |
| 72 | ILLEGAL MAXDATA SIZE | MAXDATA less than 0. |
| 80 | INSUFFICIENT STORAGE | Could not obtain enough DL area (system problem). Program is too large for the segmenter. The segmenter (SEGPROC) must be reprepared with a larger DL. |
| 81 | ILLEGAL PATCH | Bad code generated by a compiler. |
| 82 | UNABLE TO OPEN SCRATCH FILE | Scratch area used to prepare code from USL before moving to SL file. <i>numericparam</i> is FCHECK value. |
| 83 | UNABLE TO OPEN LIST FILE | FOPEN failure. <i>numericparam</i> is FCHECK value. |
| 84 | UNEXPECTED I/O ERROR | <i>numericparam</i> is FCHECK value and <i>stringparam</i> is file type. Files opened are: USL, AUX USL, SL, RL, RL LIBRARY, PROGRAM, LIST and SCRATCH files. |
| 86 | ITEM DIFFERENT FROM CLASS SPECIFICATION | One of the following occurred: <ol style="list-style-type: none"> 1. Tried to -COPY <i>item</i> and <i>item</i> class differs from class specified. 2. Tried to -PURGERBM <i>item</i> and <i>item</i> class differs from class specified. 3. Tried to -USE <i>item</i> and <i>item</i> class differs from class specified. |
| 87 | ITEM NOT PRIMARY ENTRY POINT | Tried to -ADDRL <i>item</i> and <i>item</i> not primary entry. |
| 88 | INCOMPATIBLE ITEM TYPE | One of the following occurred: <ol style="list-style-type: none"> 1. Tried to -HIDE <i>item</i> and <i>item</i> not entry point. 2. Tried to -NEWSEG <i>item</i> and <i>item</i> not procedure. 3. Tried to -PURGERBM <i>item</i> and <i>item</i> must be UNIT. |
| 89 | INVALID CLASS SPECIFICATION | One of the following occurred: <ol style="list-style-type: none"> 1. Tried to -COPY ENTRY. Must be UNIT or SEGMENT. 2. Tried to -PURGERL SEGMENT (not allowed). 3. Tried to -PURGESL UNIT (not allowed). |

Figure 10-7. Segmenter Error Messages (Continued)

| Error No. | Error Message | Comments |
|-----------|------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 93 | UNABLE TO LOCATE ITEM | One of the following occurred: <ol style="list-style-type: none"> 1. Can't find <i>item</i> in AUXUSL for -COPY. 2. Can't find <i>segname</i> in USL for -ADDSL. 3. Can't find <i>segname</i> in USL for -ADDSL. 4. Can't find <i>name</i> in USL for -HIDE and -REVEAL. 5. Can't find <i>newsegname</i> in USL for -NEWSEG. 6. Can't find <i>RLNAME</i> in RL for -PURGERL. 7. Can't find <i>name</i> in USL for -USE. |
| 110 | SEGMENT CURRENTLY LOADED | Segment referenced is in use and has been loaded by the loader. |
| 111 | SEGMENT CONTAINS EXTERNAL VARIABLE | <i>stringparam</i> is entry name. |
| 112 | SEGMENT CONTAINS COMMON | <i>stringparam</i> is entry name. |
| 113 | SEGMENT CONTAINS LOGICAL UNITS | FORTRAN program contains reference to unit numbers. |
| 120 | AUX USL FILE NOT DESIGNATED | Tried to -COPY and AUXUSL not open. |

Figure 10-7. Segmenter Error Messages (Continued)

PART 3
Optional Capabilities



✓ PH ✓

SECTION XI

Process-handling Optional Capability

As noted earlier, all user and system programs under MPE/3000 are run on the basis of *processes*—the basic executable entities in the operating system. Processes are invisible to the programmer accessing MPE/3000 through the *standard capabilities* described in Part II of this manual. This programmer has no control over processes or their structure; for him, MPE/3000 automatically creates, handles, and deletes all processes. But, the user with certain *optional capabilities* can interact with processes directly. The most basic of these optional capabilities is the *Process-Handling Optional Capability*, discussed in this section. This capability, assigned and used independently of the other optional capabilities, allows the user to

- Create and delete processes.
- Activate and suspend processes.
- Manage communication between processes.
- Change the scheduling of processes.
- Obtain information about existing processes.

These operations can be very useful to a user. For instance, they allow him to have several independent processes running concurrently on his behalf, all communicating with one another.

NOTE: In order to fully understand the information in this section, the user should first read Section II of this manual, "How MPE/3000 Operates." That section introduces background information, definitions, and concepts that are further developed in this section.

PROCESS LIFE-CYCLE

The user can activate processes to run any kind of code—programs, subroutines, or procedures. To illustrate how processes are created and managed, the steps in the life-cycle of a typical user process are discussed below and illustrated in Figure 11-1.

1. The process life-cycle begins with a source program on punched cards (for example) which is compiled (by a process running a compiler) into relocatable binary modules (RBM's) on a user subprogram library (USL) file. This USL resides on disc.

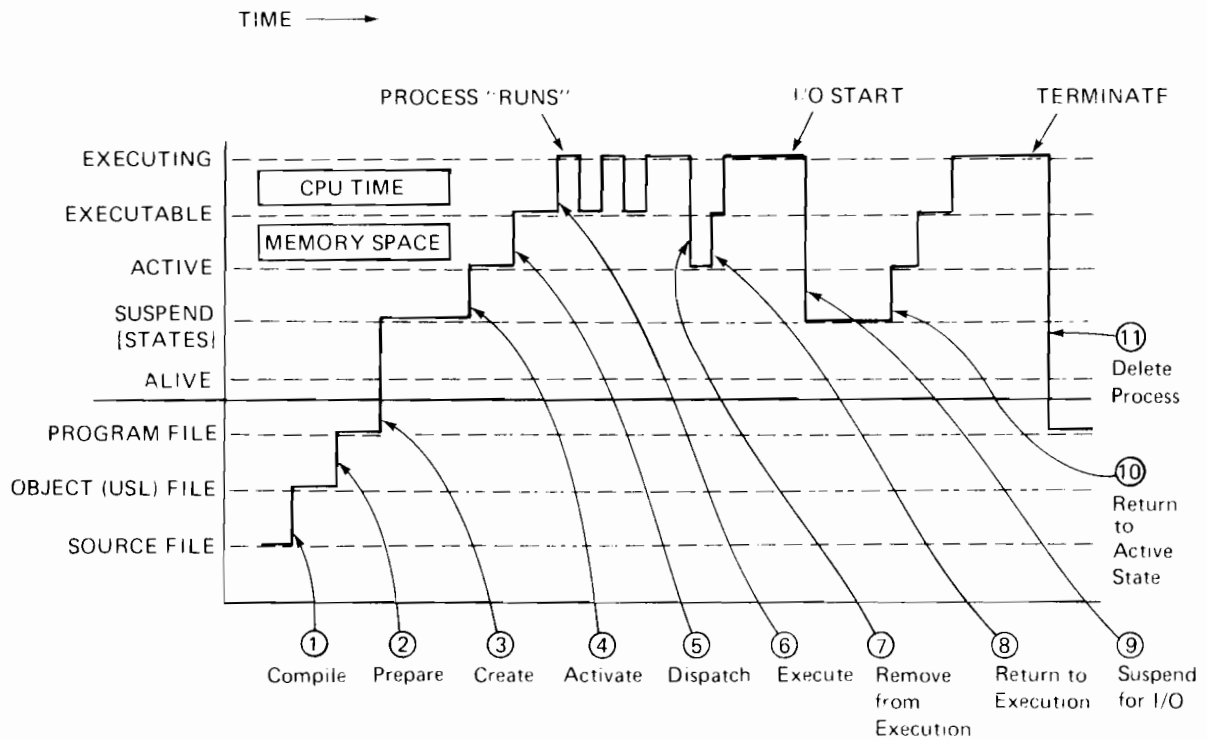


Figure 11-1. Process Life Cycle

2. The program is next prepared for execution (by a process running the MPE/3000 Segmenter). The program now resides as linked code segments on a program file which also contains the initially-defined stack for the process to be executed.
3. When an explicit or implied :RUN command is issued, allocation/execution begins with the creation of the process. The :RUN command invokes the CREATE intrinsic, which establishes a Process Control Block (PCB) entry in the PCB table, and calls the LOAD intrinsic to obtain Code Segment Table (CST) and Data Segment Table (DST) entries and virtual memory space for the process, initialize global variables, satisfy external references, and load the program into virtual memory. The LOAD intrinsic returns control to the CREATE intrinsic, which then calls the PROCREATE intrinsic to transform the virtual-memory program into a process. The PROCREATE intrinsic formats the PCB Extension, writes return and entry stack markers on the initially-defined stack, links the new process to its family, inserts the PCB entry into the master scheduling queue or subqueue according to its priority, and places the process in the *alive* state by turning on the *alive* bit in the PCB. The new process now exists, but it is not yet active and thus cannot be executed.

At this point, all the basic elements of the process exist—the PCB that defines and controls the process, the code segments that the process executes, and the data stack upon which the process operates. Within MPE/3000, the process is identified by a unique number called a process identification number (PIN). The significant tables resident in main memory (Figure 11-2) that relate to the process are

- a. *The PCB Table* contains one entry (PCB) for each process. The PCB holds information needed during the entire life of the process, whether or not it is running in main memory. When the process executes, the contents of the PCB change continually. Among other elements, the PCB contains: links to the process' father and sons, its priority, *alive* bit, and *active* bit. In addition to the PCB, the PCB Extension (physically part of the stack segment) contains process control information needed when the process is running in main memory. The PCB Extension need not be continually resident in main memory, since it may be swapped out to disc with the stack segment if the process is interrupted. The information in the PCB Extension includes the process' most recent register settings and information about files referenced by the process.
- b. *The CST* contains, for each code segment in virtual memory, a two-word entry. This entry shows the length and current address of the segment and bit-settings that indicate:
 - Whether the segment is present in or absent from main memory.
 - Whether the segment runs in privileged or non-privileged mode.
 - Whether the segment calls the TRACE routine (as distinct from the TRACE/3000 facility).
 - Reference information, used statistically by MPE/3000.

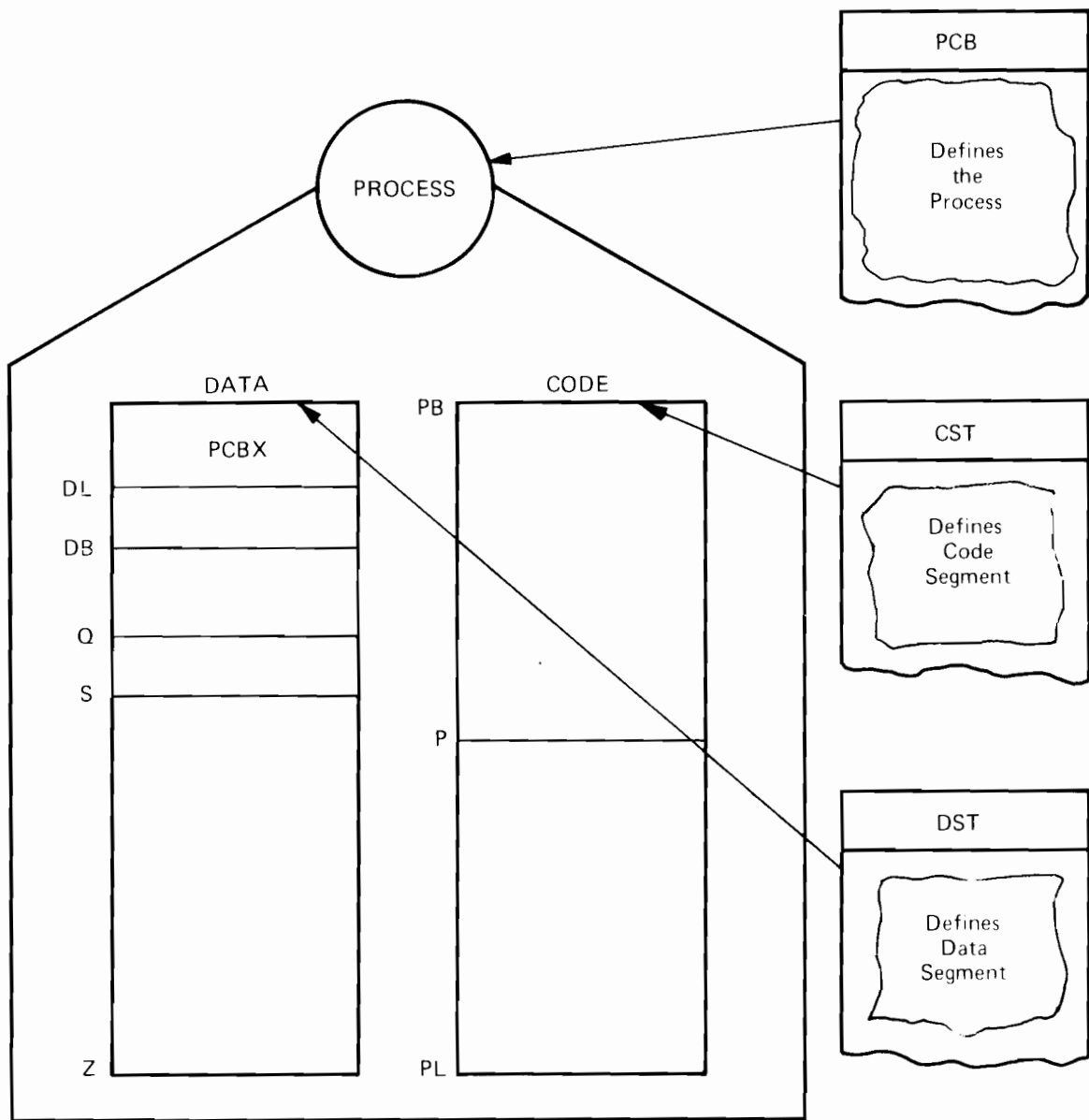


Figure 11-2. Process Components and Tables

- c. *The DST* contains, for each data segment in virtual memory, a two-word entry. This entry shows the length and current address of the segment, and bit-settings that indicate:
 - Whether the segment is present in or absent from main memory.
 - Reference information, used statistically by MPE/3000.
4. The father of this process (in this case, the job main process) calls the *ACTIVATE* intrinsic which places the process in the *active* substate by turning the active bit in the PCB *on*. Now, when the dispatcher scans the scheduling queue, it will recognize the process as active.
5. The dispatcher calls the MAPP (Make-a-Process-Present) process to move the process' data stack into main memory. This moves the process from the *active* to the *executable* state. As soon as the process becomes the highest-priority process that is both *present* and *active* in the queue, the dispatcher allocates it central processor time, sets the stack segment registers (DL,DB,Q,S and Z), and transfers control to the starting address (entry-point label) of the initial code segment.
6. The process now enters execution. From the entry-point on, it follows the sequence dictated by its code segments until its suspension or deletion. At any point, the process may call the *CREATE* intrinsic to create a son process, using a program file specified by the user for its code segments. It may then call the *ACTIVATE* intrinsic to activate the son process. At this time, the father process may suspend itself (through an implied or explicit call to the *SUSPEND* intrinsic) or may run concurrently with its son.

Figuratively speaking, a father process is responsible for what happens to its son—creation, activation, suspension, deletion, or other special operations. Each process carries a set of logical pointers linking it to other members of its family: one pointer indicates its father and one or more other pointers correspond to its sons. (All of these pointers are actually PIN's.) The pointers function as shown in Figure 11-3, with the solid lines indicating father pointers and the broken lines indicating son pointers.

7. If a new process is dispatched, the memory manager may remove the current process' stack from main memory and overlay its code segments with those of the new process (swapping). The original process is still active in virtual memory, but is no longer executable until its segments are returned to main memory.
8. When the original process' segments are returned to main memory, that process resumes execution.
9. A request for blocked input/output suspends the process while input/output is performed. In this case, the process' segments are again swapped from main memory to disc. (Note, however, that segments are not *always* swapped out when a process is suspended for input/output — this occurs here only for illustrative purposes.)
10. When input/output is complete, the process again becomes active, then executable, and then runs to completion or until the next interruption.

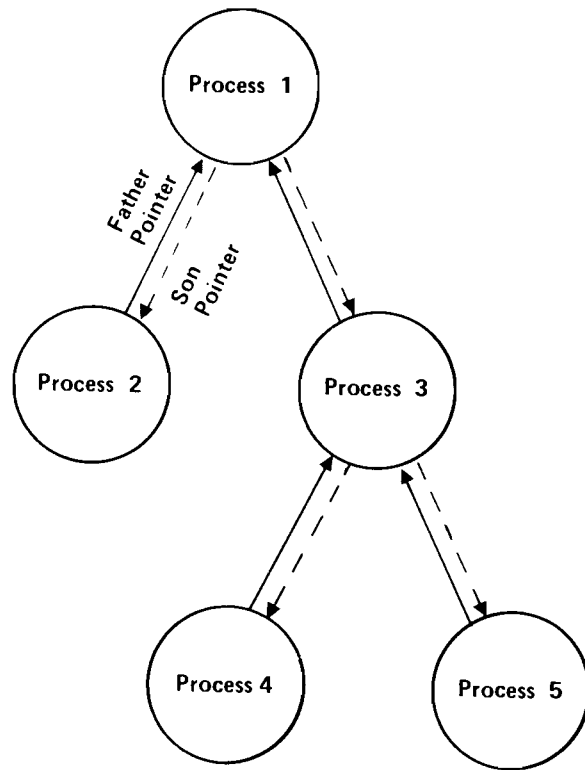


Figure 11-3. Process Linking

11. When the process is completed, it typically deletes itself by calling the `TERMINATE` intrinsic. This intrinsic, and others it invokes, accomplish the following:
 - a. Close all open files associated with the process (based on information from the PCB Extension).
 - b. Deallocate all resources, such as CST and DST entries, code and data space, and system input/output buffers.
 - c. Request the UCOP to delete the PIN, and remove the process from the process family structure and from the queue by deleting the PCB.
 - d. Turn on the *active* bit in the father process (main process, in this case) if specified by the user.

The process is now deleted from the system. The user code still exists as a program file on disc. To execute this code again, the user must create a new process.

PROCESS-HANDLING

The user creates, manages, and deletes processes with the intrinsics described below.

Creating Processes

Any running process can request the creation of a son process by issuing the `CREATE` intrinsic call, described below. The `CREATE` intrinsic loads the program to be run by the new process into virtual memory, creates the new process as the son of the calling process, initializes its data stack, schedules the process, and returns its PIN to the calling process.

The `CREATE` intrinsic is written in the following format:

```
PROCEDURE CREATE (progname,entryname,pin,param,flags,
stacksize,dlsize,maxdata,priorityclass,rank);
```

```
VALUE param,stacksize,dlsize,priorityclass,maxdata,flags,rank;
```

```
LOGICAL priorityclass,flags;
```

```
INTEGER stacksize,dlsize,maxdata,pin,param,rank;
```

```
BYTE ARRAY progname,entryname;
```

```
OPTION VARIABLE, EXTERNAL;
```


The parameters for this intrinsic call are as follows. All parameters except *progrname* and *pin* are optional.

- progrname* A byte-array containing a string, terminated by a blank, specifying the name and optionally, the account and group (*filereference* format) of the file containing the program to be run.
- entryname* A byte array, containing a string terminated by a blank, specifying the entry-point (label) in the program where execution is to begin when the process is activated. The *primary* entry-point in the program can be specified in this array by a blank character alone. If *entryname* is omitted, the primary entry-point is specified by default.
- pin* A word in which the PIN of the new process is returned to the calling process. This PIN is used in other intrinsics to reference the new process. The PIN can range from 1 to 255. (If an error is detected, a PIN of zero is returned.)
- param* A word used to transfer control information to the new process. Any instruction in the outer block of code in the new process can access this information in Location (Q-4). If *param* is omitted, this word is filled with zeros.
- flags* A word whose bits, if on, specify loading options:
- Bit (15:1) = *ACTIVE Bit*. If this bit is *on*, MPE/3000 reactivates the calling process (father) when the new process terminates. If this bit is *off*, the calling process is not activated at that time. The default setting is *off*.
 - Bit (14:1) = *LOADMAP Bit*. If this bit is *on*, a listing of the allocated (loaded) program is produced on the job/session listing device. This map shows CST entries used by the new process. If this bit is *off*, no map is produced. The default setting is *off*.
 - Bit (13:1) = *DEBUG Bit*. If this bit is *on*, a breakpoint is set at the first executable instruction of the new process. If this bit is *off*, the breakpoint is not set. The default setting is *off*. This bit is ignored if the user is non-privileged and the new process requires privileged mode. It is also ignored if the user does not have read/write access to the program file of the new process.
 - Bit (12:1) = *NOPRIV Bit*. If this bit is *on*, the program is loaded in *non-privileged mode*. If this bit is *off*, the program is loaded in the mode specified when the program file was prepared. The default setting is *off*.

Bits (10:2) = *LIBSEARCH Bits*. These bits denote the order in which libraries are to be searched for the program:

00 = System Library.

01 = Account Public Library, followed by System Library.

10 = Group Library, followed by Account Public Library, followed by System Library.

The default setting for these bits is 00.

Bits (7:3) = (Not used by MPE/3000).

Bits (5:2) = *STACKDUMP Bits*. These bits control the arming/disarming of the mechanism by which the stack is dumped in the event of an abort:

00 = Arms only if armed at father level.

01 = Arm unconditionally.

10 = Same as 00.

11 = Disarm unconditionally for new process.

The default setting for these bits is 00.

NOTE: The following bits (0:4) are used only when the preceding bit-pair (5:2) is 01. Otherwise, these bits are ignored.

Bit (4:1) = (Not used.)

Bit (3:1) = *DL to QI Bit*. If this bit is *on*, the portion of the stack from DL to QI is dumped. If this bit is *off*, this portion is not dumped. The default setting is *off*.

Bit (2:1) = *QI to S Bit*. If this bit is *on*, the portion of the stack from QI to S is dumped. If this bit is *off*, this portion is not dumped. The default setting is *off*.

Bit (1:1) = *Q-63 to S Bit*. If this bit is *on*, the portion of the stack from Q-63 to S is dumped. If this bit is *off*, this portion is not dumped. The default setting is *off*.

Bit (0:1) = *ASCII DUMP Bit*. If this bit is *on*, the dump is interpreted in ASCII, in addition to the octal dump. If this bit is *off*, ASCII interpreting is not given. The default setting is *off*.

If *flags* omitted, all default values noted are taken.

| | |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| stacksize | An integer (Z-Q) denoting the number of words assigned to the local stack area (the area bounded by the initial Q and Z registers). The default value is that specified in the program file. |
| dlsiz | An integer (DB-DL) denoting the number of words in the user-managed stack area (bounded by the DL and DB registers). The default value is that specified in the program file. |
| maxdata | The maximum size allowed for the process' stack (Z-DL) area in words. When specified, this value overrides the one established at program-preparation time. The default value is specified at system generation time. |
| priorityclass | A string of two ASCII characters describing the priority class (subqueue) in which the new process is scheduled; this may be: "AS", "BS", "CS", "DS", or "ES", as described in the discussion <i>Rescheduling Processes</i> later in this section. (For users with other optional capabilities, the possible entries also include "AS" and "BS".) The default value is the priority of the calling process. (See p 14-6) |
| rank | The relative rank of the process within a linear subqueue. (This parameter is available only to users with the <i>System Supervisor Capability</i> , as discussed in <i>MPE/3000 Operating System, System Manager/System Supervisor Manual</i> .) (See p 8-7 ↗) |

NOTE: For the stacksize, dlsiz, and maxdata parameters, a value of -1 indicates that the MPE/3000 Segmenter is to assign default values; specifying -1 is equivalent to omitting the parameter.

When the CREATE intrinsic is called, the DB register must be pointing at the user's stack.

The CREATE intrinsic returns one of the following condition codes to the calling process:

| | |
|-----|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CCE | Request was granted; the new process is created. |
| CCG | Request was granted; the maxdata and/or dlsiz parameters given were illegal, but other values were used. Specifically, <ol style="list-style-type: none"> 1. If the maxdata specified exceeds that maximum Z-DL allowed by the configuration, then that configuration maximum value is assigned. 2. If (dlsiz + 100) modulo 128 is not zero, then dlsiz is rounded upward so that (dlsiz + 100) modulo 128 = 0. |
| CCL | The request was not granted because the proname or entryname specified does not exist. |

The creating process is aborted if:

1. Request was rejected because of illegal parameters; a PIN of zero is returned. Specifically, this occurs:
 - If *progrname* is illegal.
 - If *entryname* is illegal.
 - If *stacksize* is less than 512 (decimal) and is not -1. (Note than -1 specifies the default value.)
 - If *dlsiz*e is less than 0 and is not -1.
 - If *maxdata* is less than or equal to 0, and is not -1.
 - If (*dlsiz*e + *globsize* + *stacksize* + 128) exceeds *maxdata*. Note that *dlsiz*e may have been modified to satisfy Condition 2, under CCG, described above. (The *globsize* value is the sum of the primary DB plus the secondary DB values — the total DB given at program preparation time by the program map (PMAP).)
 - If (*dlsiz*e + *globsize* + *stacksize* + 128) exceeds the maximum *stacksize* defined during system configuration. Note that *dlsiz*e may have been modified to satisfy Condition 2, under CCG, described above.
 - If (*maxdata* + 90) exceeds 32768, where *maxdata* is either the value passed as a parameter or a value re-computed by the Loader under Condition 1 of CCG (above).
2. The user does not have the *Process-Handling Optional Capability*.
3. An illegal value (a non-existent subqueue) was specified for the *priorityclass* parameter.
4. A required parameter (*progrname* or *pin*) is omitted.
5. A reference parameter was not within the required range.



EXAMPLE:

Suppose the user wants to create a process that, when activated, runs a program in the file *PROGA*, beginning at the entry-point *START1*. He also wants the libraries searched in the following order: *account public library*, followed by *system library*. The process' PIN is to be returned to the word *PINRET*. The process is to be assigned the *priorityclass CS*. The user enters the following intrinsic call. (The byte array *PROG* contains "PROGA", the byte array *ENTER* contains "START1", and the tenth bit of the word *FLAGGER* is on.)

```
CREATE (PROG,ENTER,PINRET,,FLAGGER,,,,"CS");
```

The PIN assigned to the processes is 026, returned to the word *PINRET*.

Activating Processes

After a process has been created, it must be activated in order to run. When this is done, the process runs until it is suspended or deleted. A newly-created process can only be activated by its father. A process that has been suspended (with the SUSPEND intrinsic call discussed later) can be reactivated by its father or any of its sons (as specified in the SUSPEND call *susp* parameter). To activate or reactivate a process, the ACTIVATE call is issued:

PROCEDURE *ACTIVATE* (*pin*,*susp*) ;

VALUE pin,susp;

LOGICAL susp;

INTEGER pin;

OPTION VARIABLE, EXTERNAL;

The ACTIVATE call parameters are

pin An integer specifying the PIN of the calling process' father or son to be activated. (A non-zero integer is used for sons, and zero is used for the father.) In order for the ACTIVATE intrinsic to work, the receiving process must be expecting the activation signal from the calling process (as noted in the discussion of the SUSPEND intrinsic.)

susp A word that specifies whether or not the calling process is to be suspended when the called process is activated. When *susp* is not present or is set zero, the calling process remains active. When *susp* is specified, the calling process is suspended. The 14th and 15th bits of *susp* specify the anticipated source of the call that later will reactivate the calling process.

Bit (15:1) = If *on*, the process expects to be activated by its father.

Bit (14:1) = If *on*, the process expects to be activated by one of its sons.

If both of these bits are on, the suspended process can be activated by either father or sons.

Bits (0:14) = Not usable by programmers with *only* the *Process-Handling Optional Capability*.

MPE/3000 guarantees that no interrupts occur between activation of the called process and suspension of the calling process.

The following condition codes can be returned.

CCE Request granted; Process *pin* is activated. The calling process is suspended if *susp* was specified.

CCG Process *pin* is already active. The calling process is suspended if *susp* was specified.

CCL Request rejected, because Process *pin* was not expecting activation by this calling process; an illegal *pin* parameter was specified; or the *susp* parameter was not valid.

The process is aborted if:

1. The user does not have the *Process-Handling Optional Capability*.
2. A required parameter is omitted.
3. A job or session Main Process, or a System Process, is specified.

EXAMPLE:

To activate the process whose PIN is stored in *PINVAL*, but without suspending itself, the calling process issues:

ACTIVATE (PINVAL);

Suspending Processes

A process can suspend itself by issuing the *SUSPEND* intrinsic call. When this is done, the process relinquishes its access to the central processor until reactivated by an *ACTIVATE* intrinsic call. When it suspends itself, the process must specify the anticipated source of this *ACTIVATE* call (its father or son process). When the process is reactivated, it begins execution with the instruction immediately following the *SUSPEND* call. The format of the *SUSPEND* call is

PROCEDURE *SUSPEND (susp, rin) ;*

VALUE *susp, rin;*

LOGICAL *susp;*

INTEGER *rin;*

OPTIONAL VARIABLE, EXTERNAL;

The parameters are

susp A word whose 14th and 15th bits specify the anticipated source of the call that later will reactivate the process. For processes run by users with only the *Process-Handling Optional Capability*, at least *one* of these bits must be *on*.

Bit (15:1) = If *on*, the process expects to be activated by its father.

Bit (14:1) = If *on*, the process expects to be activated by one of its sons.

If both of these bits are *on*, the suspended process can be activated by either father or sons.

Bits (0:14) = (These bits are not used by users with only the *Process-Handling Optional Capability*.)

rin A RIN designation, (defined in Section IX). If *rin* is specified, it represents a *local* RIN locked by the process but released when the process is suspended. This facility can be used to synchronize processes within the same job. If *rin* is omitted, no RIN is unlocked.

The following condition codes can be returned:

CCE Request granted.
CCG (Not returned.)
CCL Request rejected, because of invalid parameter (*susp* not valid, *RIN* not owned by session/job, or *RIN* not locked.)

The process is aborted if the user issuing the SUSPEND call does not have the Process-Handling Capability.

EXAMPLE:

The following intrinsic call suspends the calling process, which then expects to be reactivated by its father. (The source of reactivation is specified by setting bit 15 of the word PARENT on.) No RIN's are involved.

SUSPEND (PARENT)

Deleting Processes

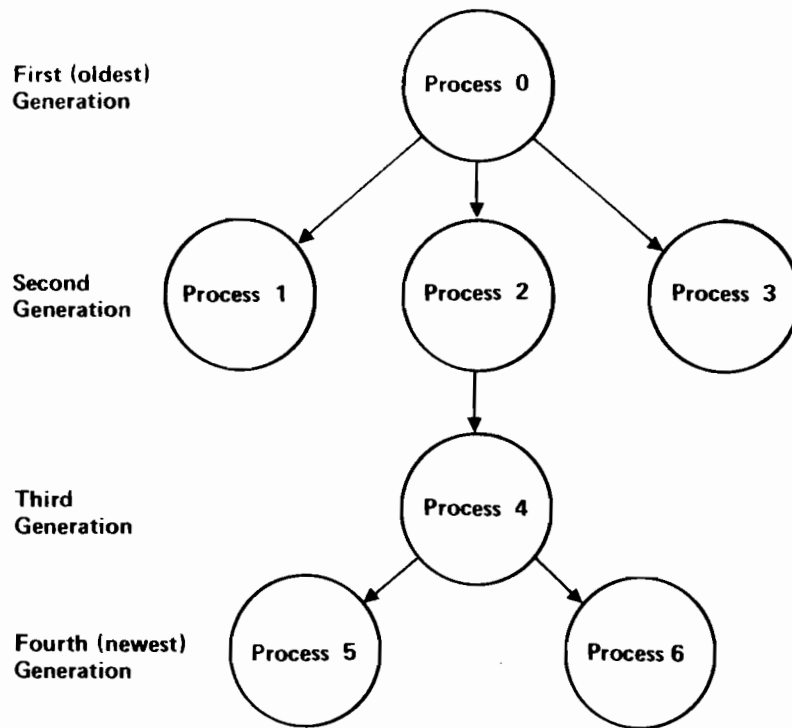
A process can request the deletion of itself, or any of its sons at any time. When this is done, all code and data segments in the process, and all resources owned by the process, are released; all temporary files opened by the process are closed; and finally, the PIN is released. When a process is deleted, MPE/3000 also automatically deletes all descendants of that process, as shown in Figure 11-4. Within a process tree structure, the newest generations are deleted first. Within each generation, processes are deleted in order of their creation.

A process deletes itself by issuing the TERMINATE intrinsic call.

In a job or session main process, the TERMINATE intrinsic is automatically invoked by detection of an end-of-job/session condition. This intrinsic removes the job or session from the system.

The format of the TERMINATE call is

```
PROCEDURE TERMINATE ;  
OPTION EXTERNAL;
```



(When Process 2 is deleted, the following structure results.)

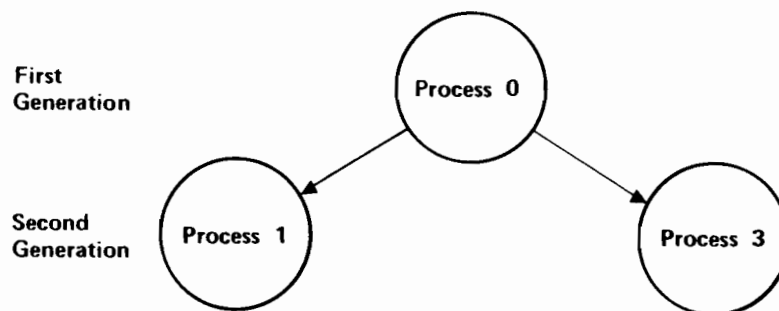


Figure 11-4. Process Deletion

A process can delete one of its sons by issuing the KILL intrinsic call

```
PROCEDURE KILL (pin) ;  
VALUE pin;  
INTEGER pin;  
OPTION EXTERNAL;
```

In this call, *pin* is a word containing the PIN of the process to be deleted; it can be an integer value ranging from 1 to 255.

The following condition codes can be returned by the KILL intrinsic:

| | |
|-----|-----------------------------------------------------------------|
| CCE | Request granted. |
| CCG | Request granted; the specified process was already terminating. |
| CCL | Request denied because an illegal PIN was specified. |

Interprocess Communication

The user can direct the communication of information between processes. This information transfer, however, is restricted to upward or downward paths through the process tree structure, so that any process can communicate only with its father or sons. Between any father/son pair, only one such transfer is allowed at any particular time.

Information transferred between processes is referred to as *mail*. It is sent from one process to another through an intermediate storage area called a *mailbox*. At any given time, a mailbox can contain only one item of mail (a *message*). For any process, there are two sets of mailboxes:

- The mailbox used for communication between the process and its father (each process has one of these)
- The set of mailboxes used for communication between the process and its sons (each process has one of these mailboxes for each of its sons)

The transmittal of mail is based upon a transaction between the sending and receiving processes that involves the following steps:

1. Optionally, the sending process tests the mailbox to determine its status (whether it is empty, contains a message, or is being used by the receiving process).
2. The sending process transmits the mail to the mailbox. (The message transferred is a word array in the sending process' stack, defined by a starting location and word count. The smallest message allowed is a single word. MPE/3000 automatically performs a *bounds check* that ensures that the array specified actually falls within the limits of the process' stack.)
3. The receiving process optionally tests the mailbox to determine its status.

4. If the mailbox contains a message, the receiving process collects this mail. If the mail is not collected, it is overwritten by additional mail from the sending process. (When the mail is collected, another bounds check is performed to validate the address given for the stack of the receiving process.)

Testing Mailbox Status

A process can determine the status of the mailbox used by its father or son, with the MAIL intrinsic call. If the mailbox contains mail that is awaiting collection by this process, the length of this message (in words) is returned to the calling process: this enables the calling process to initialize its stack in preparation for receipt of the message. The MAIL intrinsic call format is

LOGICAL PROCEDURE

| |
|---------------------|
| MAIL (pin, count) ; |
|---------------------|

VALUE pin;

INTEGER pin, count;

OPTION EXTERNAL;

This intrinsic returns as the value of MAIL, the *status* of the mailbox, as noted below.

The parameters for this call are

- pin* An integer specifying the mailbox tested. If this integer specifies the mailbox of a son process, it must be the PIN of that son. If it specifies the mailbox of a father process, it must be zero.
- count* A word to which an integer denoting the *length* (in words) of any incoming mail in the mailbox is to be returned.

The mailbox status returned to the calling process (as the value of MAIL) will be one of the following digits:

| Status Returned | Meaning |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------|
| 0 | The mail box is empty. |
| 1 | The mailbox contains previous <i>outgoing</i> mail from this calling process that has not yet been collected by the destination process. |
| 2 | The mailbox contains <i>incoming</i> mail awaiting collection by this calling process; the length of the mail is found in <i>count</i> . |
| 3 | An error occurred because an invalid <i>pin</i> was specified or a bounds check failed. |
| 4 | The mailbox is temporarily inaccessible because other intrinsics are using it in the preparation or analysis of mail. |

In addition to the mailbox status code, the MAIL intrinsic returns the following condition codes:

- CCE Request granted; the mailbox status was tested.
- CCG Request denied because an illegal *pin* parameter was specified (MAIL has the value of 3).
- CCL (Not returned by this intrinsic.)

If the user does not have the *Process-Handling Capability*, the calling process is aborted.

EXAMPLE:

To test the status of the mailbox associated with one of its son processes, (whose PIN is stored in the word SONNY), a process issues the following call:

STATCOUNT := MAIL (SONNY, MCOUNT);

Because the mailbox contained an incoming message 10 words long, the integer 10 is stored in the word MCOUNT, and the status code 2 is stored in the word STATCOUNT.

Sending Mail

A process sends mail to its father or sons by issuing the SENDMAIL intrinsic call. If the mailbox for the receiving process contains a message sent previously by the calling process but not collected by the receiving process, the action taken depends upon the *waitflag* parameter specified in SENDMAIL. If the mailbox is currently being used by other intrinsics, the SENDMAIL intrinsic waits until the mailbox is free and then sends the mail.

The SENDMAIL intrinsic call format is

LOGICAL PROCEDURE *SENDMAIL (pin, count, location, waitflag) ;*

VALUE pin, count, waitflag;

INTEGER pin, count;

LOGICAL waitflag;

ARRAY location;

OPTION EXTERNAL;

This intrinsic returns, as the value of SENDMAIL, one of the status codes noted later in this discussion.

The call parameters are

| | |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>pin</i> | An integer specifying the process to receive the mail. If a son process is specified, the integer is the PIN of that process. If a father process is specified, the integer is zero. |
| <i>count</i> | An integer specifying the length of the message, in words transmitted from the sending process' stack. If zero is specified, SENDMAIL empties the mailbox of any previous incoming or outgoing mail. |
| <i>location</i> | The starting address of the array containing the message in the sending process' stack (relative to the address in the DB register). |
| <i>waitflag</i> | A word specifying (in Bit 15) the action to be taken if the mailbox contains previously-sent mail: TRUE = Wait until the receiving process collects the previous mail before sending current mail. (Bit 15 = 1) FALSE = Cancel (overwrite) any previously-sent mail with the current mail. (Bit 15 = 0) |

The SENDMAIL intrinsic returns one of the following status codes to the user's program.

| Status Returned | Meaning |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0 | The mail was successfully transmitted; the mailbox contained no previous mail. |
| 1 | The mail was successfully transmitted; the mailbox contained previously-sent mail that was overwritten by the new mail, or contained previous incoming/outgoing mail that was cleared. |
| 2 | The mail was not successfully transmitted because the mailbox contained incoming mail to be collected by the sending process (regardless of the <i>waitflag</i> setting). |
| 3 | An error occurred because an illegal <i>pin</i> was specified, or a bounds-check failed. |
| 4 | An illegal wait request would have produced a deadlock. |
| 5 | The request was rejected because <i>count</i> exceeded the maximum mailbox size allowed by the system. |
| 6 | The request was rejected because storage resources for the mail data segment were not available. |

The SENDMAIL intrinsic returns one of the following condition codes:

- CCE Request granted; the mail was sent.
- CCG Request not granted because of an illegal *count* parameter (SENDMAIL has the value of 5), or an illegal *pin* (SENDMAIL has the value of 3), or storage for the mail data segment was not available (SENDMAIL has the value of 6).
- CCL Request not granted because the bounds-check revealed that the *count* or *location* parameters did not define a legal stack area (SENDMAIL has the value of 3) or both processes are waiting to send mail (SENDMAIL has the value of 4).

If the user does not have the *Process-Handling Capability*, the process is aborted.

EXAMPLE:

To send mail (defined in the stack by an array called LOCAT, and 3 words long) to its father, a process issues the following call. If the mailbox contains a previously-sent message, that message will be overwritten. (The word CNTR contains 3, and Bit 15 of the word NOWAIT is set off).

```
STAT := SENDMAIL (0,CNTR,LOCAT,NOWAIT);
```

Because the mailbox contained no previous mail, and the sender's mail was successfully transmitted, the value 0 is stored in the word STAT.

Receiving (Collecting) Mail

A process collects mail transmitted to it (from its father or son) by calling the RECEIVEMAIL intrinsic. If the mailbox for the receiving process is empty, the action taken depends on the *waitflag* parameter specified in RECEIVEMAIL. If the mailbox is currently being used by other intrinsics, the RECEIVEMAIL intrinsic waits until the mailbox is free before accessing it.

The RECEIVEMAIL intrinsic call is written in the following format:

```
LOGICAL PROCEDURE RECEIVEMAIL (pin,location,waitflag) ;
```

```
VALUE pin,waitflag;
```

```
INTEGER pin;
```

```
LOGICAL waitflag;
```

```
ARRAY location;
```

```
OPTION EXTERNAL;
```

This intrinsic returns, as the value of `RECEIVEMAIL`, the mailbox status as noted below.

The call parameters are

- pin* An integer specifying the process sending the mail. If a son process is specified, the integer is the PIN of that process. If a father process is specified, the integer is zero.
- location* The starting address of the word array in the receiving process' stack where the collected message is to be written. (This address is relative to the address in the DB register.)
- waitflag* A word specifying the action to be taken if the mailbox is empty:
- TRUE = Wait until incoming mail is ready for collection.
 (Bit 15 = 1)
- FALSE = Return immediately to the calling process. (Bit 15 = 0)

The status code indicating the result of the `RECEIVEMAIL` intrinsic will be one of the following:

| Status Returned | Meaning |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0 | The mailbox was empty (and the <i>waitflag</i> parameter was FALSE). |
| 1 | No message was collected because the mailbox contained outgoing mail from the receiving process. |
| 2 | The message was successfully collected. |
| 3 | An error occurred because of an illegal pin or bounds check failure. |
| 4 | The request was rejected because <i>waitflag</i> specified that the receiving process wait for mail if the mailbox is empty, but the other process sharing the mailbox is already suspended, waiting for mail. If <i>both</i> processes were suspended, neither could activate the other, and they may be <i>deadlocked</i> . |

The `RECEIVEMAIL` intrinsic call returns one of the following condition codes:

- CCE Request granted; the mail was collected.
- CCG Request not granted because of illegal *pin* parameter (`RECEIVEMAIL` has the value of 3).
- CCL Request not granted because the bounds check revealed that the *location* parameter did not define a legal stack address (`RECEIVEMAIL` has the value of 3) or because both sending and receiving processes would be awaiting incoming mail (deadlock). (`RECEIVEMAIL` has the value of 4).

If the user does not have the *Process-Handling Capability*, the process is aborted.

EXAMPLE:

To collect a message from a son process (whose PIN is contained in P2) and store this message in the stack array MDATA, a process could issue the following call. If the mailbox is empty, the RECEIVEMAIL intrinsic waits until a message arrives. (Bit 15 of the word WAIT is set on.)

```
STAT := RECEIVEMAIL (P2,MDATA,WAIT);
```

Because the mail is successfully collected, the mailbox status code 2 is returned to the word STAT.

Avoiding Deadlocks

Since the simultaneous use of mail-transmission, process suspension, and RIN-locking intrinsics throughout a process structure could result in a deadlock if the intrinsic calls are not properly synchronized, the user should be aware of the following facts:

1. In a multi-process job/session, whenever a process is suspended (through the SUSPEND intrinsic or when locking a RIN or receiving mail), MPE/3000 does not determine whether all other processes in the tree are suspended. The user must exercise caution in avoiding such a situation.
2. An attempt by a process to lock a global RIN will succeed only if two conditions are met:
 - a. No other process within the job/session has currently locked this RIN—a *global* RIN cannot be used as a *local* RIN, because deadlock within the same job/session could otherwise occur.
 - b. The calling process currently has no other global RIN locked for itself; this could otherwise result in deadlock between two job/sessions.

Rescheduling Processes

When a process is created, it is scheduled on the basis of a priority class assigned by its father. After this point, its priority class can be changed at any time through the GETPRIORITY intrinsic call. (A process can change its own priority or that of a son but it cannot reschedule its father.)

Generally, MPE/3000 schedules processes in linear or circular subqueues, as described in Section II. The standard linear subqueues are

- The *AS* subqueue, containing processes of very high priority.
- The *BS* subqueue, containing processes of high priority.
- The *ES* subqueue, containing idle processes with low priority.

The circular subqueues are

- The *CS* subqueue, actually composed of two sub-subqueues, used for interactive and multiprogramming batch processes.
- The *DS* subqueue, also composed of two sub-subqueues, available for general use at a lower priority than the *CS* subqueue.

The subqueue to which a process belongs determines the priority class of the process. From highest to lowest priority, these classes (named after their subqueues), are

AS
BS
CS
DS
ES

The format of the *GETPRIORITY* intrinsic call is

```
PROCEDURE GETPRIORITY (pin,priorityclass,rank);  
VALUE pin,priorityclass,rank;  
LOGICAL priorityclass;  
INTEGER pin,rank;  
OPTION VARIABLE, EXTERNAL;
```


The parameters for the GETPRIORITY intrinsic are

- pin* An integer denoting the process whose priority class is to be changed. If this is a son process, the integer is the process' PIN. If this is the calling process, the integer is zero.
- priorityclass* A string of two ASCII characters describing the priority class (subqueue) in which the process is rescheduled; this may be "AS", "BS", "CS", "DS", or "ES". For users with other optional capabilities, this may be any subqueue or portion of the master queue permitted by those capabilities. (A process can be scheduled in the master queue by specifying xA , where x is an actual priority number.) (SEE p 14-6)
- rank* The relative rank of the process within a linear subqueue. (This parameter is available only to users with the *System Supervisor Capability*, as discussed in *MPE/3000 Operating System, System Manager/System Supervisor Manual*.)

The GETPRIORITY intrinsic returns one of the following condition codes:

- CCE Request granted.
- CCG The process is not alive.
- CCL Request denied because an illegal PIN was specified.

The process is aborted if the user does not have the Process-Handling Capability or specifies a non-existent priority class (subqueue).

EXAMPLE:

To reschedule itself with the priority class "ES," a process issues the following call:

GETPRIORITY (0,"ES");

Determining Source of Activation

After a suspended process is reactivated, it can determine whether the source of the activation request was its father process or one of its son processes. It does this by issuing the GETORIGIN call:

```
INTEGER PROCEDURE GETORIGIN ;  
OPTION EXTERNAL;
```

This call returns either of the following codes (as the value of GETORIGIN):

| Code | Meaning |
|------|---------------------|
| 1 | Activated by father |
| 2 | Activated by a son |

The condition code is not changed by this intrinsic.

Determining Father Process

A process can determine the PIN of its father by issuing the FATHER intrinsic call.

```
INTEGER PROCEDURE FATHER ;  
OPTION EXTERNAL;
```

This call returns the PIN to the calling process (as the value of FATHER).

In addition, one of the following condition codes is returned to the calling process:

| | |
|-----|---------------------------------------------------------------|
| CCE | Request granted, the father is a user process. |
| CCG | Request granted; the father is a job or session main process. |
| CCL | Request granted; the father is a system process. |

Determining Son Processes

A process can request the return of the PIN assigned to any of its sons, with the GETPROCID intrinsic call:

```
INTEGER PROCEDURE GETPROCID (son) ;  
VALUE son;  
INTEGER son;  
OPTION EXTERNAL;
```

The PIN is returned as the value of GETPROCID.

The *son* parameter designates the specific son in chronological terms—in other words, the *son-th* son still in existence. If *son* exceeds the number of sons currently attached to the calling process, *zero* is returned as the value of GETPROCID.

The condition code is not changed by this intrinsic.

EXAMPLE:

The following command returns the PIN of the sixth existing son of the calling process. (The word VAL contains the integer 6.) The PIN is transmitted to the word PINNO.

```
PINNO := GETPROCID (VAL);
```

Determining Process Priority and State

A process can request the return of a two-word message denoting the following information about its father or sons:

| Word | Bits | Meaning |
|------|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1 | (8:8) | The process' priority number in the master queue. |
| | (0:8) | Not used by MPE/3000; these bits are set to zero by the system. |
| 2 | (15:1) | Activity state. If <i>on</i> , the process is active; if <i>off</i> , the process is suspended. |
| | (13:2) | Suspension condition (set only if bit 15 is <i>off</i>); the <i>on</i> -bit indicates the source of the expected activation: Bit 14 = Father Bit 13 = Son |
| | (9:4) | Not used by MPE/3000; these bits are set to zero by the system. |
| | (7:2) | The origin of the last ACTIVATE intrinsic call, where 01 = Father and 10 = Son. If the value is <i>zero</i> , the process was activated by MPE/3000. |
| | (4:3) | Queue Characteristics: 001 = Master queue. 111 = Linear subqueue 100 = Circular subqueue |
| | (0:4) | Not used by MPE/3000; these bits are set to zero by the system. |

The above information is returned with the GETPROCINFO intrinsic call:

```

DOUBLE PROCEDURE GETPROCINFO (pin);
VALUE pin;
INTEGER pin;
OPTION EXTERNAL;

```

The double-word information described above is returned as the value of GETPROCINFO.

The parameter is

pin The process to which the returned message pertains. If this is a father process, *pin* is zero. If it is a son process, *pin* is the PIN of that process.

The GETPROCINFO call returns the following condition codes:

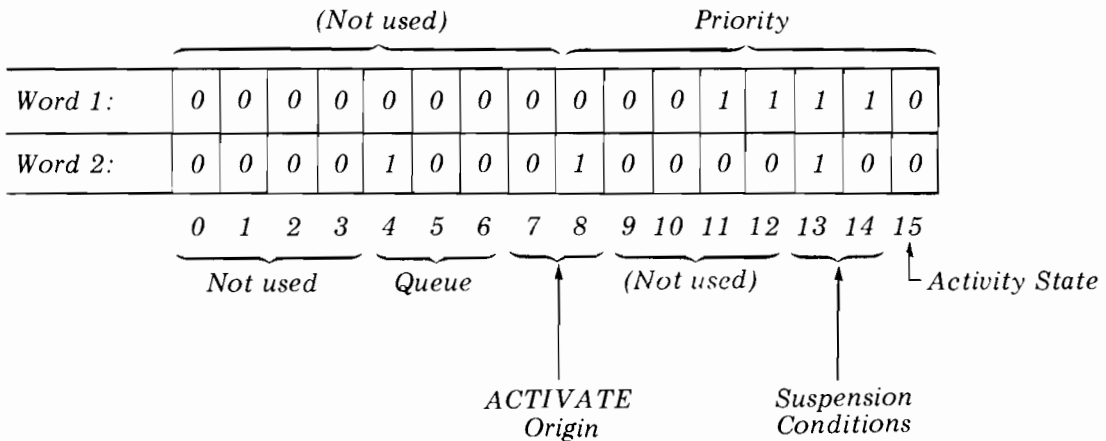
- CCE Request granted.
- CCG Request not granted because the process is being terminated.
- CCL Request not granted because an illegal PIN was specified.

EXAMPLE:

To request information about its father, a process issues the following call:

```
INFO := GETPROCINFO (0);
```

The information returned to the double-word INFO is



This information is interpreted as follows, (regarding the father process).

| Word 1: | Bit Nos. | Value | Meaning |
|---------|----------|-------|-----------------------------------------------------------------|
| | (8:8) | 30 | <i>Process has priority 32 in master queue.</i> |
| | (0:8) | 0 | <i>(Not used.)</i> |
| Word 2: | (15:1) | 0 | <i>Process is suspended.</i> |
| | (14:1) | 0 | } <i>Process to be activated by its son.</i> |
| | (13:1) | 1 | |
| | (9:4) | 0 | <i>(Not used.)</i> |
| | (7:2) | 1 | <i>Origin of last ACTIVATE call was father of this process.</i> |
| | (4:3) | 4 | <i>Circular subqueue.</i> |





SECTION XII

Data-segment Management Optional Capability

In Section II, it was noted that each process possesses a private data segment that contains the data generated and manipulated by the process. In a user process, this segment is referred to as the user's stack segment, and can serve many different purposes.

MPE/3000 also allows users with the *Data-Segment Management Optional Capability* to create and access extra data segments for their processes during a job or session. These segments are used for temporary storage while the creating processes exist. Additionally, each segment is assigned an identity that either allows it to be shared between different processes in a job or session, or declares it private to the calling process. When a process terminates, all private data segments created by it are automatically destroyed. Sharable data segments are saved until explicitly deleted or until the job or session ends, at which point they are destroyed. Extra data segments are not directly addressable by user processes. They can only be accessed through intrinsics that move data between the user's stack and the extra data segments by switching the pointer in the DB-Register. If a process not assigned the *Data-Segment Management Capability* attempts to call these intrinsics, that process is aborted.

The maximum number of extra data segments allowed per process is determined at system configuration time, but is never more than four.

CREATING AN EXTRA DATA SEGMENT

A process can create or acquire an extra data segment by issuing the GETDSEG intrinsic call. (The number of extra data segments that can be requested, and the maximum size allowed these segments, are limited by parameters specified when the system is configured.) When an extra data segment is created, the GETDSEG intrinsic returns to the calling process a *logical index number*, assigned by MPE/3000, that allows this process to reference the segment in later intrinsic calls. The GETDSEG intrinsic is also used to assign the segment the *identity* (noted previously) that either allows other processes in the job or session to share the segment, or that declares it private to the calling process. If the segment is sharable, other processes can obtain its logical index (through GETDSEG) and use this index to reference the segment. Thus, the logical index is a local name that identifies the segment throughout any process that obtained the index with the GETDSEG call. (The logical index need not be the same value in all processes sharing the data segment.) The *identity*, on the other hand, is a job-wide or session-wide name that permits any process to determine the logical index of the segment.

The format of the GETDSEG intrinsic call is

```
PROCEDURE GETDSEG (index,length,id);  
VALUE id;  
LOGICAL index,id;  
INTEGER length;  
OPTION EXTERNAL;
```

The parameters are

index A word to which the logical index of the data segment, assigned by MPE/3000, is returned.

length The size of the data segment (if the segment is not yet created) or the word to which the size of the segment is returned (if the segment already exists).

id A word containing the identity that declares the data segment sharable between other processes in the job, or private to the calling process. For a sharable segment, *id* is specified as a non-zero value. (If a data segment with the same *id* already exists, it is made available to the calling process. Otherwise, a new data segment, sharable within the job/session, is created with this *id*.) For a private data segment, an *id* of zero (0) should be specified.

The following condition codes may be returned:

CCE Request granted; a new segment was created.

CCG Request granted; an extra data segment with this identity already existed.

CCL Request denied because an illegal length was specified (INDEX has the value of %2000); the process requested more than the maximum allowable number of data segments (INDEX has the value of %2001), or sufficient storage was not available for the data segment (INDEX has the value of %2002).

EXAMPLES:

To create a new data segment with the identity specified in *XSEG* and a length of 200 words (specified in the word *LNPTH*), the user enters the following call. The logical index is returned to the word *LI*.

```
GETDSEG (LI, LNPTH, XSEG);
```

To determine the logical index and length of an existing extra data segment with the identity specified in *ASEG*, a process enters this next call. The logical index is returned to the word *IND* and its length to the word *LEN*.

```
GETDSEG (IND, LEN, ASEG);
```

DELETING AN EXTRA DATA SEGMENT

A process can release an extra data segment assigned to it by issuing the *FREEDSEG* intrinsic call. If this is a private data segment, or if it is a sharable segment not currently assigned to any other process in the job/session, the segment is deleted from the entire job/session. Otherwise, it is deleted from the calling process but continues to exist in the job/session.

```
PROCEDURE FREEDSEG (index, id);
```

```
VALUE index, id;
```

```
LOGICAL index, id;
```

```
OPTION EXTERNAL;
```

The intrinsic call parameters are

index A word containing the logical index assigned to the data segment, obtained from the *GETDSEG* intrinsic call.

id The identity (if any) assigned to the segment. If none is assigned, zero (0) should be entered.

The following condition codes may be returned:

- CCE Request granted; the data segment is deleted from the job.
- CCG Request granted; the data segment is deleted from process but continues to exist in the job.
- CCL Request denied; either the *index* is invalid, or *index* and *id* do not specify the same extra data segment.

EXAMPLE:

To delete a data segment with the logical index contained in *INDX*, the user enters the following call. Because the segment has no *id*, it is deleted from both the process and the job.

FREEDSEG (INDX,0)

TRANSFERRING DATA FROM EXTRA DATA SEGMENT TO STACK

A process can copy a block of words from an extra data segment into the stack by issuing the *DMOVIN* intrinsic call. A bounds check is performed by the intrinsic on both the extra data segment and the stack to ensure that the data is taken from within the data segment boundaries and moved to an area within the stack boundaries.

PROCEDURE *DMOVIN (index,disp,number,location);*

VALUE index,disp,number;

LOGICAL index;

INTEGER disp,number;

ARRAY location;

OPTION EXTERNAL;

The parameters are

| | |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>index</i> | A word containing the logical index of the extra data segment, obtained from the GETDSEG call. |
| <i>disp</i> | The displacement of the first word in the block to be transferred, from the first word in the data segment. (This must be an integer value greater than or equal to zero.) |
| <i>number</i> | The size of the data block to be transferred, in words. (This must be an integer value greater than or equal to zero.) |
| <i>location</i> | The starting location of the array (buffer) in the stack (relative to the DB address) where the data block is to be moved. |

The following condition codes may be returned:

| | |
|-----|----------------------------------------------------------------------------|
| CCE | Request granted. |
| CCG | Request denied because of bounds-check failure. |
| CCL | Request denied because of illegal <i>index</i> or <i>number</i> parameter. |

EXAMPLE:

To transfer a block of data 64 words long from an extra data segment whose logical index is specified by LOGX, to the stack, a process can issue to the following call. The data block begins at the tenth word (disp=9) in the data segment. It is entered in the stack beginning at the starting address specified in STDATA.

DMOVIN (LOGX,9,64,STDATA);

TRANSFERRING DATA FROM STACK TO EXTRA DATA SEGMENT

A process can copy a block of words from the stack to an extra data segment through the DMOVOUT intrinsic call. A bounds check is initiated to ensure that the data is taken from an area within the stack boundaries and moved to an area within the extra data segment boundaries.

PROCEDURE

DMOVOUT (*index,disp,number,location*);

VALUE index,disp,number;

LOGICAL index;

INTEGER disp,number;

ARRAY location;

OPTION EXTERNAL;

The parameters are

| | |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>index</i> | A word containing the logical index of the extra data segment, obtained through the GETDSEG call. |
| <i>disp</i> | The displacement, in the extra data segment, of the first word of the receiving buffer from the first word in the data segment. (This must be an integer greater than or equal to zero.) |
| <i>number</i> | The size of the data block to be transferred, in words. (This must be an integer greater than or equal to zero.) |
| <i>location</i> | The starting address of the data block in the stack (relative to the DB address). |

The following condition codes can be returned:

| | |
|-----|----------------------------------------------------------------------------|
| CCE | Request granted. |
| CCG | Request denied because of bounds-check failure. |
| CCL | Request denied because of illegal <i>index</i> or <i>number</i> parameter. |

EXAMPLE:

To transfer a block of data 32 words long from the stack to an extra data segment whose logical index is specified in LOGY, a process can issue the following call. The data block begins at the address in the stack specified in GODATA. The receiving buffer begins at the first word in the extra data segment (disp=0).

DMOVOUT (LOGY,0,32,GODATA);

Changing Size of Data Segment

The user can alter the current size of an extra data segment by calling the ALTDSEG intrinsic. As a typical application, after the user has obtained disc storage for a new segment by calling GETDSEG, he may issue ALTDSEG to reduce the storage required by the segment when it is moved into main memory, and later expand it as needed, for more efficient use of memory. (In no case, can ALTDSEG be used to increase segment size beyond that originally assigned through GETDSEG.) The ALTDSEG intrinsic format is

```
PROCEDURE ALTDSEG (index,inc,size);
```

```
VALUE index, inc;
```

```
LOGICAL index;
```

```
INTEGER inc,size;
```

```
OPTION EXTERNAL;
```

The parameters are

- index* A word containing the logical index of the extra data segment, obtained through the GETDSEG call.
- inc* The value (in words) by which the data segment is to be changed. A positive integer value requests an increase, and a negative integer value denotes a decrease.
- size* A word to which is returned the new size of the data segment after incrementing or decrementing takes place.

The following condition codes can be returned:

- CCE Request granted.
- CCG Request not fully granted; an illegal decrement, requesting a new total segment size of 0 or less, or an illegal increment, requesting a new size greater than the size originally assigned by GETDSEG, was attempted. In the first case, the current size remains in effect; in the second case, the original size is granted (and this size is returned through the *size* parameter).
- CCL Request denied because an illegal *index* parameter was specified.



MR

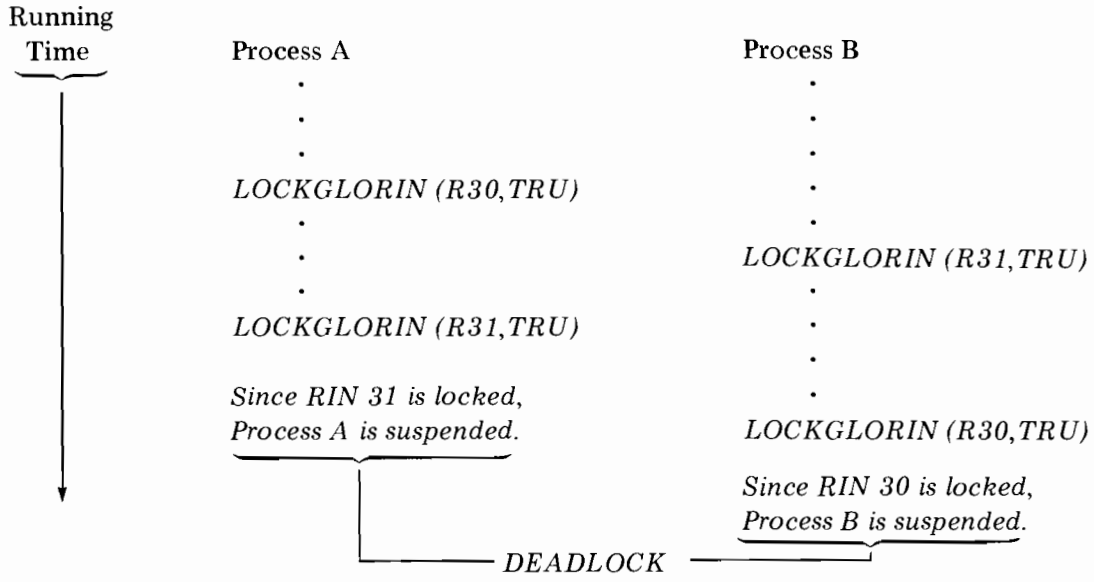
SECTION XIII

Multiple Resource Identification Number Optional Capability

As discussed in Section IX, users can manage the sharing of resources (such as input/output devices, files, programs, or procedures) between jobs or processes through resource identification numbers (RIN's). Users having only *Standard MPE/3000 Capabilities* can lock only one global or local RIN at a time. However, users having the *Multiple RIN Optional Capability* can lock as many global RIN's as desired simultaneously, with no checking by MPE/3000. (This applies to the FLOCK intrinsic for files as well as the LOCKGLORIN intrinsic for global RIN's.) Under multiprogramming, this capability introduces a risk that deadlocking may occur unless the users cooperating in resource management are very careful. In deadlocking, two jobs or processes that have been suspended cannot be resumed because they are mutually blocked. The following example illustrates how deadlocking can occur.

EXAMPLE:

In this example, Process A and Process B, running concurrently, are managing two RINs, 30 and 31. At one point, Process A locks RIN 30 (designated by R30). Subsequently, Process B locks RIN 31 (designated by R31). Still later, Process A tries to lock RIN 31 while it is still locked by Process B. Thus, Process A is suspended. Process B then tries to lock RIN 30, still locked by Process A. As a result, Process B is also suspended. Since Process A has RIN 30 locked and needs RIN 31, and Process B has RIN 31 locked and needs RIN 30, both processes are blocked and cannot become unblocked; they are deadlocked.



As a guide in avoiding deadlocks, users preparing jobs or processes that cooperate in RIN management can logically pre-order their use of RINs in the same way. For example, Job A and Job B can both issue requests to lock RINs in ascending numerical order (RIN's 10, 11, 12, and so forth). This ensures that neither job ever attempts to lock a RIN of lower logical rank than any RIN currently locked, and a deadlock never occurs. The RIN's can be released in any order.



SECTION XIV

Privileged Mode Optional Capability

Normally, an MPE/3000 user can access only his own code and data areas in main memory. But a user with the *Privileged Mode Optional Capability* can access all areas of the system and can use all features of the hardware. This user can access all normally-uncallable intrinsics and system tables. He can invoke all system instructions, including those in the privileged central processor instruction set. He can, in short, use the computer on the same terms as MPE/3000 itself.

CAUTION: No hardware mechanism is provided to prevent a program running in privileged mode from damaging the environments of other users or destroying MPE/3000 itself. For such a program, the only bounds-checking performed is that for stack overflow (where the S register contains an address greater than that in the Z register). Thus, a privileged-mode program constitutes a potential hazard to the system. When preparing and running such programs, users should be very cautious, particularly regarding the locations referenced by the program.

A programmer with the *Privileged Mode Optional Capability* can use this capability in two ways:

1. He can write permanently privileged programs that are loaded and executed entirely in privileged mode.
2. He can write temporarily privileged programs that dynamically enter and leave privileged mode during execution, as required.

PERMANENTLY PRIVILEGED PROGRAMS

A program's segments are loaded and executed directly in privileged mode when all three of the following conditions exist:

1. Any of the program's code segments contains privileged instructions.
2. The program is prepared with the *Privileged Mode Optional Capability*, by entering the appropriate capability-class attribute in the *caplist* parameter of the :PREP or :PREPRUN command that prepares the program. This implies that the *user* issuing this command must himself have been assigned the *Privileged Mode Optional Capability*.
3. The NOPRIV optional parameter is omitted from the :PREPRUN or :RUN command that executes the program, or the CREATE intrinsic that creates a process to run it. (This omission turns the privileged mode bit in the appropriate CST entries *on*.)

When a user adds a segment to a Segmented Library (through the -ADDSL Segmenter Command), the procedures within the segment are checked to determine if any one of them is privileged. If it is, the segment is always run in privileged mode. (In order to add a segment containing one or more privileged procedures to a library, the user must himself possess the *Privileged-Mode Optional Capability*.)

TEMPORARILY PRIVILEGED PROGRAMS

Temporarily privileged programs are initiated, upon request, in the non-privileged mode. Then, intrinsics can be issued to change the program to and from the privileged mode dynamically. For example, just before a set of privileged instructions is encountered, the program can be switched to the privileged mode to allow execution of these instructions. Then, after the last privileged instruction in the set is encountered, the program can be returned to non-privileged mode. This bracketing of privileged instructions aids in reducing system violations, since the program cannot access locations or resources outside the user's environment when it is running in non-privileged mode.

The user running a temporarily-privileged program should understand how the central processor handles procedure calls (PCAL instructions) and exits (EXIT instructions) when encountered in either mode:

In the privileged mode, when a PCAL instruction is executed, privileged mode is retained even though the destination code segment may have a non-privileged CST entry. When an EXIT instruction is encountered, the resulting mode depends upon the status word in the stack marker.

In the non-privileged mode, when a PCAL instruction is encountered, the mode assumed is obtained from the CST entry for the destination code segment. When an EXIT instruction occurs, the resulting mode is taken from the CST entry indicated in the stack marker. If this entry indicates privileged mode, a system violation occurs.

In general, the status word determines the action taken in privileged mode, but the CST entry determines the action in non-privileged mode.

A program is loaded and begins execution as a temporarily privileged program (in the non-privileged mode) when these two conditions are met:

1. The program is prepared with the *Privileged Mode Optional Capability*, by entering the appropriate capability-class attribute in the *caplist* parameter of the :PREP or :PREPRUN command that prepares the program. This implies that the user issuing this command must himself have been assigned the *Privileged Mode Optional Capability*.
2. The NOPRIV optional parameter is *included* in the :PREPRUN or :RUN command that executes the program, or the CREATE intrinsic that creates a process to run it.

When a temporarily-privileged program is initiated, the CST entries corresponding to its segments have their privileged-mode bits set *off*.

The intrinsics for dynamically switching modes are described below.

ENTERING PRIVILEGED MODE

To switch a temporarily-privileged program from the non-privileged mode to the privileged mode, the GETPRIVMODE intrinsic call is issued. This intrinsic turns the privileged mode bit in the status register *on*, but leaves the privileged mode bit in the CST entry for the executing segment *off*. Thus, if additional segments are to be run as part of the program, they will be run in privileged mode unless an intrinsic is specifically called to return to the non-privileged mode, because the status register rather than the CST determines a mode change when running in privileged mode. The GETPRIVMODE call is written simply as

```
PROCEDURE GETPRIVMODE;  
  
OPTION EXTERNAL;
```

The following condition codes may be returned:

- CCE Request granted; the program was in non-privileged mode when the intrinsic call was issued.
- CCG Request granted; the program was already in privileged mode when the intrinsic call was issued.
- CCL (Not returned by this intrinsic.)

The calling process is aborted if the user does not possess the *Privileged Mode Optional Capability*, and the CST indicates non-privileged mode.

ENTERING NON-PRIVILEGED MODE

To change a temporarily-privileged program from the privileged to the non-privileged mode, the GETUSERMODE intrinsic call is issued. This intrinsic changes the privileged mode bit in the status register to *off*. The intrinsic call is written simply as

```
PROCEDURE GETUSERMODE;  
OPTION EXTERNAL;
```

The following condition codes may be returned:

- CCE Request granted; the program was in privileged mode when the intrinsic call was issued.
- CCG Request granted; the program was already in non-privileged mode when the intrinsic call was issued.
- CCL (Not returned by this intrinsic.)

MOVING THE DB-POINTER

When a user with the *Data Segment Management Optional Capability* runs a process with an extra data segment in privileged mode, he can prepare for movement of data between this segment and the stack by calling the SWITCHDB intrinsic. This intrinsic changes the DB register so that it points to the base of the extra data segment rather than the base of the stack. This intrinsic returns the logical index of the data segment indicated by the previous DB register setting, allowing the user to restore this setting later. (If the previous DB setting indicated the stack, *zero* is returned.)

USER MODE

LOGICAL PROCEDURE

SWITCHDB (*index*);

VALUE *index*;

LOGICAL *index*;

OPTION PRIVILEGED, EXTERNAL;

The previous DB-register setting is returned as the value of SWITCHDB.

The *index* parameter denotes the logical index of the data segment to which the DB register is switched, as obtained through the GETDSEG intrinsic call. (MPE/3000 checks the value specified for *index*, to ensure that the process has previously acquired access to this segment.) For an extra data segment, the *index* must be a positive, non-zero integer. To switch back to the stack segment, the *index* must be zero.

The following condition codes can be returned by the SWITCHDB intrinsic:

- CCE Request granted.
- CCG (Not returned by this intrinsic.)
- CCL Request rejected, because an illegal data segment index was referenced.



The calling process is aborted if the user does not have the *Privileged Mode Optional Capability*.

EXAMPLE:

To set the DB register so that it points to the base of an extra data segment whose logical index is indicated in the word INDEX2, this intrinsic call is issued. (The previous DB-register setting is returned to the word SET.)

```
SET := SWITCHDB (INDEX2);
```

OTHER DATA-SEGMENT INTRINSICS

Users with the *Privileged-Mode Optional Capability* can also call all intrinsics available to users with the *Data Segment Management Optional Capability* (Section XII), provided that they acknowledge these rules:

1. When calling the data segment intrinsics from *privileged mode*, ensure that the DB register points to its normal stack position. When GETDSEG is used to create extra data segments under these conditions, the number of segments that can be created is limited only by the space available in the process Control Block Extension; this is virtually unlimited.
2. When a temporarily-privileged process calls a data segment intrinsic while in *non-privileged mode*, the data segment index returned to the calling process can also be used by the process to reference that segment in *privileged mode*. But, if the process calls a data segment intrinsic in *privileged mode*, the index returned *cannot* be used to reference the segment in *non-privileged mode*.

SCHEDULING PROCESSES

A user with the *Privileged Mode Optional Capability* can schedule processes in areas of the master scheduling queue that are not available to other users.

The master queue (Figure 14-1) is divided into logical areas, each corresponding to a particular type of dispatching and priority class for the processes within it. A logical area can be a linear subqueue, a circular subqueue, or a portion of the main master queue. In a linear subqueue, the process with highest priority accesses the central processor first and maintains this access until the process either is completed, terminated, or suspended to await the availability of a required resource. In a circular subqueue, all processes have equal priority and each accesses the central processor for an interval (time quantum) of maximum duration (or until completed, terminated, or suspended). At the end of this duration, control is transferred to the next process in the queue, continuing in a round-robin fashion. This time-slicing is controlled by the system timer. Processes that are not scheduled in a subqueue are scheduled in the master queue.

Each subqueue in the master queue is defined by a priority number. While the standard user is aware of the priority class associated with a subqueue, only a user with *Privileged Mode Capability* or a System Supervisor User is concerned with priority numbers. The standard subqueues (priority classes) are as follows.

AS is a linear subqueue containing processes of very high priority. (In a subsequent MPE/3000 release, it will be accessible only to users with a special *MPE/3000 Optional Capability* to be added to the system then.) Its priority number is 30.

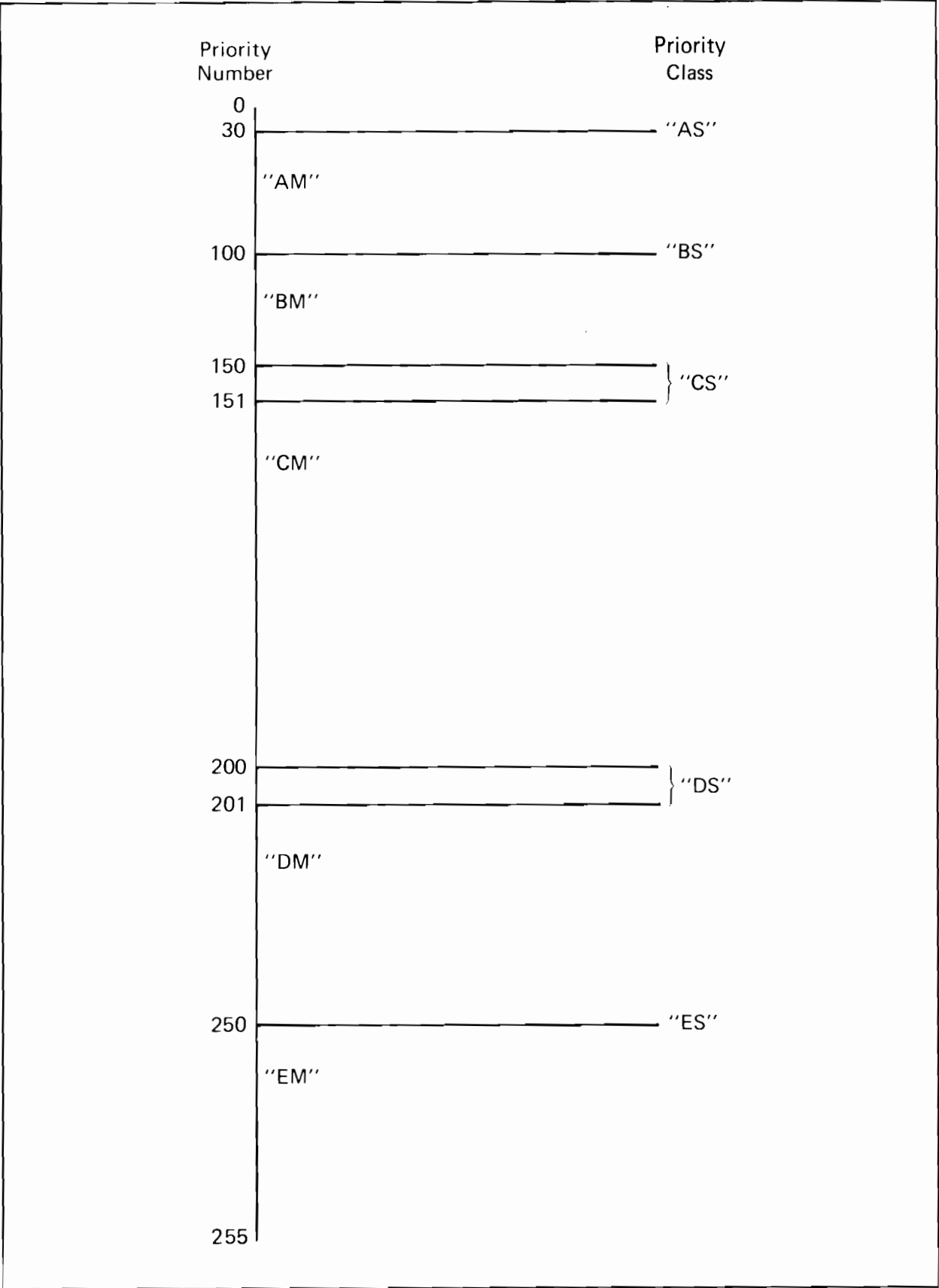


Figure 14-1. MPE/3000 Master Queue Structure

| | |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| BS | is a linear subqueue containing processes of high priority. (In a subsequent MPE/3000 release, it will be accessible only to users having a special <i>MPE/3000 Optional Capability</i> to be added to the system). Its priority number is 100. |
| CS | is composed of two circular subqueues, each having a time-quantum determined at system generation time. This subqueue is used mainly for interactive processes and for multiprogramming batch jobs. The time quantum specifies the maximum execution time allowed any process in the subqueue before going on to the next process. The priority numbers of the subqueues are 150 and 151. |
| DS | is also composed of two circular subqueues, each having a time-quantum determined at configuration time. This subqueue is available for general use at a lower priority than the CS subqueue. The priority numbers of the subqueues are 200 and 201. |
| ES | is a linear subqueue, primarily containing idle processes with low priority. Its priority number is 250. |
| AM,BM,CM,DM,EM | are discontinuous portions of the main master queue. Processes can be scheduled in these areas only if they are system processes or are initiated by a user with Privileged Mode or System Supervisor Capability. Such processes are dispatched linearly. |

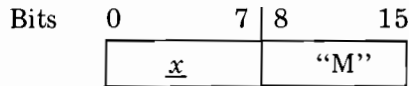
The priority class of a process can be specified by the normal user with standard or optional MPE/3000 capabilities. In the two-character string that comprises a priority-class reference, the first character refers to the location of a subqueue within the master queue (in alphabetical order) and the second character specifies whether the logical area is the subqueue itself (S) or the portion of the master queue (M) that immediately follows the subqueue. When a priority class is requested for a process, it is assigned the lowest priority within that class (relative to other processes already assigned the same class). In a circular subqueue, the actual priority of the process is modified as other processes use central processor time. In a linear subqueue, an automatic adjustment is made as necessary.

Only a user with Privileged Mode Capability can assign a priority number to a process. Priority numbers range from 1 to 255 inclusively, with 1 denoting the highest priority. Any two processes having the same priority number are in the same subqueue. The privileged mode user also can specify the relative ranks of processes having identical numbers within a linear subqueue, and can assign them specific priorities in the master queue.

With each circular subqueue of priority P is associated another subqueue of higher priority. (As noted above, these are the subqueues within the CS and DS classifications.) Normally, users requesting CS or DS priority are assigned the lower subqueue in the classification requested. However, users with *Privileged Mode Optional Capability* can access the higher subqueue by specifying priority numbers as noted below.

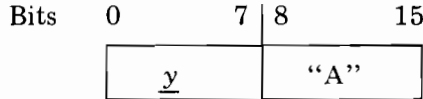
Priorities are assigned to processes through the *priorityclass* parameter of the CREATE and/or GETPRIORITY intrinsic (Section XI). Because users with the Privileged Mode Capability can schedule processes within the master queue, the *priorityclass* parameter can take on the following values:

1. A string of two ASCII characters describing the standard priority-class (subqueue) in which the process is to be scheduled; the standard subqueues are "AS", "BS", "CS", "DS", or "ES".
2. An ASCII character (*x*) specifying a valid subqueue, followed immediately by the single-character string "M", indicating the Master Queue. The word format is:



This schedules the process in that region of the master queue directly adjacent to and below the subqueue *x*. The process is scheduled in the first available priority in that region.

3. A number (*y*) specifying an absolute priority number, followed by the single-character string "A" indicating that *y* is an absolute priority number. The word format is:



This schedules the process at the location specified by *y* in the Master Queue. If another process or subqueue already occupies the location designated by *y*, the process is placed in the first location available moving toward the ES subqueue, and the process priority number is changed to reflect that location.

PRIVILEGED MODE DEBUG EXTENSIONS

Privileged users may use all of the DEBUG capabilities described in Section VIII plus the various privileged capabilities described below. Since privileged users may access absolute quantities and system data bases, and are not constrained by security, integrity, or bounds checking, *caution should be observed in using these extended capabilities.*

Upon invoking DEBUG in privileged mode, the message returned is of this format:

* $\left\{ \begin{array}{l} \text{DEBUG} \\ \text{BREAK} \end{array} \right\}$ * [SYS.]PRIV. nextlocation

SYS. Indicates System breakpoint mode (see A command below)

PRIV. Indicates privileged mode of operation.

nextlocation Indicates the location of the next instruction to be executed upon return.

A sixth error code, in addition to the five listed earlier for non-privileged users, is possible:

| Code | Meaning |
|------|------------------------------------------------------------------------|
| SAME | Conflict between System and Private breakpoints (see A command below). |

Privileged mode extensions consist of:

- Three additional commands (A, F, U), defined first below.
- Additional capabilities for some of the previously defined commands; only the new capabilities are discussed for these, following the A, F, U command definitions.

Switching Breakpoint Modes

The A command ("Accept" breakpoint mode) permits the privileged user to switch between Private and System breakpoint modes by entering:

?AS (switch to System breakpoint mode)
?AP (switch to Private breakpoint mode)

These modes are defined as follows:

- Private: Breakpoints are private to the current process domain and have no effect on other processes. Many processes may share the same breakpoint location under this mode.
- System: Breakpoints are global, and directed toward whichever process is first to execute the breakpoint instruction. Breakpoints set for the system console (or any system process) are automatically system breakpoints. It is not possible for private and system mode to share the same location.

Freezing a Segment in Memory

The F command permits the privileged user to freeze an absolute code segment or data segment in main memory by entering:

$$?F \left\{ \begin{array}{l} CO \\ DA \end{array} \right\} expression$$

CO Indicates code segment.

DA Indicates data segment.

expression Expression indicating segment number.

EXAMPLE:

?FDA22 (*freezes data segment 22*)

Unfreezing a Segment in Memory

The U command permits the privileged user to unfreeze a segment previously frozen by the F command, by entering:

$$?U \left\{ \begin{array}{l} CO \\ DA \end{array} \right\} expression$$

CO, *DA*, *expression* are as defined above for the F command.

EXAMPLE:

?UC0123 (*unfreezes code segment 123*)

Extensions to B Command

Privileged users can set breakpoints in absolute code segments (as well as logical code segments) by preceding the segment parameter with the letter A. Also, they can set breakpoints in system library segments by preceding the segment parameter with the letter S.

A B@ command while in System breakpoint mode will display all breakpoints belonging to the system.

EXAMPLE:

The following B command establishes a breakpoint at location 76 of absolute code segment 45.

```
?B A45.76
```

Extensions to C Command

In System breakpoint mode, a C@ command clears all system owned breakpoints.

Extensions to L Command

The privileged user may switch display output to a specific logical device.

EXAMPLE:

```
?L 14 (switches display to logical device 14)
```

Extensions to DR Command

If all registers are to be displayed (by omitting parameter list after command) the following values are also displayed:

| | |
|------|---------------------------------------------------|
| PCB | Process Control Block index |
| CST | Absolute code segment index |
| STAK | Stack segment index |
| DST | Extra data segment index |
| DX | Current value of DB-register if in absolute mode. |

Extensions to D Command

Possible display bases (*dispbases*) include the following:

| | |
|----|---------------------------------------------------------|
| A | Absolute relative (base = location 0) |
| SY | System global relative (base = system base) |
| CO | Code segment relative (base = base of any code segment) |
| DA | Data segment relative (base = base of any data segment) |
| DX | Current absolute DB relative (base = absolute DB) |
| V | Virtual memory (special; see the following page). |

The command for virtual memory display takes the form:

?DV [logdev] [+low-order sector address] [:high-order sector address] [,sector count]

logdev Logical device number; if omitted, system disc is implied.

Indirection is applied relative to the given base. Indirect addressing is not possible for the virtual memory display base.

A restriction for the CO, DA, and V display bases is that expressions must be enclosed in parentheses (see "factor" definition under the heading "DEBUG Command Format" in Section VIII).

EXAMPLES:

| <i>D Command</i> | <i>Item Displayed</i> |
|----------------------------|-------------------------------------------------------------------------|
| <i>?DA 22*2,2</i> | <i>Displays descriptor for code segment 22</i> |
| <i>?DV+120,2</i> | <i>Displays two sectors from system disc starting at sector 120</i> |
| <i>?DA 1:46*2+1:-10,10</i> | <i>Displays memory links for data segment 46 assuming it is present</i> |

Extensions to M Command

Possible modification bases (*modbase*) include the following:

| | |
|----|---------------------------------------------------------|
| A | Absolute relative (base = location 0) |
| SY | System global relative (base = system base) |
| DA | Data segment relative (base = base of any data segment) |
| DX | Current absolute DB relative (base = absolute DB) |

EXAMPLES:

| | |
|--------------------|--------------------------------------------------------------------------------------------------------------|
| <i>?MSY54</i> | <i>Modify relative location 54 in system global area</i> |
| <i>?MDA26+5:,2</i> | <i>Modify two cells in data segment 26 starting at the location pointed to be location 5 in the segment.</i> |

Extension to T Command

In privileged mode, the T command also displays the absolute CST number.



PART 4
Appendices



APPENDIX A

ASCII Character Set

| Character | Octal Value (Left Byte) | Octal Value (Right Byte) | Meaning |
|-----------|----------------------------|-----------------------------|----------------------------------------------|
| NUL | 000000 | 000000 | Null (Normally ignored by MPE/3000) |
| SOH | 000400 | 000001 | Start of heading |
| STX | 001000 | 000002 | Start of text |
| ETX | 001400 | 000003 | End of text |
| EOT | 002000 | 000004 | End of transmission |
| ENQ | 002400 | 000005 | Enquiry |
| ACK | 003000 | 000006 | Acknowledge |
| BEL | 003400 | 000007 | Bell |
| BS | 004000 | 000010 | Backspace |
| HT | 004400 | 000011 | Horizontal tabulation |
| LF | 005000 | 000012 | Line feed (Normally ignored by MPE/ 3000) |
| VT | 005400 | 000013 | Vertical tabulation |
| FF | 006000 | 000014 | Form feed |
| CR | 006400 | 000015 | Carriage return |
| SO | 007000 | 000016 | Shift out |
| SI | 007400 | 000017 | Shift in |
| DLE | 010000 | 000020 | Data link escape |
| DC1 | 010400 | 000021 | Device control 1 |
| DC2 | 011000 | 000022 | Device control 2 |
| DC3 | 011400 | 000023 | Device control 3 |
| DC4 | 012000 | 000024 | Device control 4 |
| NAK | 012400 | 000025 | Negative acknowledge |
| SYN | 013000 | 000026 | Synchronous idle |
| ETB | 013400 | 000027 | End of transmission block |

(Normally
ignored by
MPE/3000)

| Character | Octal Value (Left Byte) | Octal Value (Right Byte) | Meaning |
|-----------|----------------------------|-----------------------------|---------------------|
| CAN | 014000 | 000030 | Cancel |
| EM | 014400 | 000031 | End of medium |
| SUB | 015000 | 000032 | Substitute |
| ESC | 015400 | 000033 | Escape |
| FS | 016000 | 000034 | File separator |
| GS | 016400 | 000035 | Group separator |
| RS | 017000 | 000036 | Record separator |
| US | 017400 | 000037 | Unit separator |
| SP | 020000 | 000040 | Space |
| ! | 020400 | 000041 | Exclamation point |
| ” | 021000 | 000042 | Quotation mark |
| # | 021400 | 000043 | Number sign |
| \$ | 022000 | 000044 | Dollar sign |
| % | 022400 | 000045 | Percent sign |
| & | 023000 | 000046 | Ampersand |
| , | 023400 | 000047 | Apostrophe |
| (| 024000 | 000050 | Opening parenthesis |
|) | 024400 | 000051 | Closing parenthesis |
| * | 025000 | 000052 | Asterisk |
| + | 025400 | 000053 | Plus |
| , | 026000 | 000054 | Comma |
| - | 026400 | 000055 | Hyphen (Minus) |
| . | 027000 | 000056 | Period (Decimal) |
| / | 027400 | 000057 | Slant |
| 0 | 030000 | 000060 | Zero |
| 1 | 030400 | 000061 | One |
| 2 | 031000 | 000062 | Two |
| 3 | 031400 | 000063 | Three |
| 4 | 032000 | 000064 | Four |
| 5 | 032400 | 000065 | Five |
| 6 | 033000 | 000066 | Six |
| 7 | 033400 | 000067 | Seven |

| Character | Octal Value (Left Byte) | Octal Value (Right Byte) | Meaning |
|-----------|----------------------------|-----------------------------|---------------|
| 8 | 034000 | 000070 | Eight |
| 9 | 034400 | 000071 | Nine |
| : | 035000 | 000072 | Colon |
| ; | 035400 | 000073 | Semi-colon |
| < | 036000 | 000074 | Less than |
| = | 036400 | 000075 | Equals |
| > | 037000 | 000076 | Greater than |
| ? | 037400 | 000077 | Question mark |
| @ | 040000 | 000100 | Commercial at |
| A | 040400 | 000101 | Uppercase A |
| B | 041000 | 000102 | Uppercase B |
| C | 041400 | 000103 | Uppercase C |
| D | 042000 | 000104 | Uppercase D |
| E | 042400 | 000105 | Uppercase E |
| F | 043000 | 000106 | Uppercase F |
| G | 043400 | 000107 | Uppercase G |
| H | 044000 | 000110 | Uppercase H |
| I | 044400 | 000111 | Uppercase I |
| J | 045000 | 000112 | Uppercase J |
| K | 045400 | 000113 | Uppercase K |
| L | 046000 | 000114 | Uppercase L |
| M | 046400 | 000115 | Uppercase M |
| N | 047000 | 000116 | Uppercase N |
| O | 047400 | 000117 | Uppercase O |
| P | 050000 | 000120 | Uppercase P |
| Q | 050400 | 000121 | Uppercase Q |
| R | 051000 | 000122 | Uppercase R |
| S | 051400 | 000123 | Uppercase S |
| T | 052000 | 000124 | Uppercase T |
| U | 052400 | 000125 | Uppercase U |
| V | 053000 | 000126 | Uppercase V |
| W | 053400 | 000127 | Uppercase W |

| Character | Octal Value (Left Byte) | Octal Value (Right Byte) | Meaning |
|-----------|----------------------------|-----------------------------|-----------------|
| X | 054000 | 000130 | Uppercase X |
| Y | 054400 | 000131 | Uppercase Y |
| Z | 055000 | 000132 | Uppercase Z |
| [| 055400 | 000133 | Opening bracket |
| \ | 056000 | 000134 | Reverse slant |
|] | 056400 | 000135 | Closing bracket |
| ^ | 057000 | 000136 | Circumflex |
| _ | 057400 | 000137 | Underscore |
| · | 060000 | 000140 | Grave accent |
| a | 060400 | 000141 | Lowercase a |
| b | 061000 | 000142 | Lowercase b |
| c | 061400 | 000143 | Lowercase c |
| d | 062000 | 000144 | Lowercase d |
| e | 062400 | 000145 | Lowercase e |
| f | 063000 | 000146 | Lowercase f |
| g | 063400 | 000147 | Lowercase g |
| h | 064000 | 000150 | Lowercase h |
| i | 064400 | 000151 | Lowercase i |
| j | 065000 | 000152 | Lowercase j |
| k | 065400 | 000153 | Lowercase k |
| l | 066000 | 000154 | Lowercase l |
| m | 066400 | 000155 | Lowercase m |
| n | 067000 | 000156 | Lowercase n |
| o | 067400 | 000157 | Lowercase o |
| p | 070000 | 000160 | Lowercase p |
| q | 070400 | 000161 | Lowercase q |
| r | 071000 | 000162 | Lowercase r |
| s | 071400 | 000163 | Lowercase s |
| t | 072000 | 000164 | Lowercase t |
| u | 072400 | 000165 | Lowercase u |
| v | 073000 | 000166 | Lowercase v |
| w | 073400 | 000167 | Lowercase w |

| Character | Octal Value (Left Byte) | Octal Value (Right Byte) | Meaning |
|-----------|----------------------------|-----------------------------|-----------------------|
| x | 074000 | 000170 | Lowercase x |
| y | 074400 | 000171 | Lowercase y |
| z | 075000 | 000172 | Lowercase z |
| { | 075400 | 000173 | Opening (left) brace |
| | 076000 | 000174 | Vertical line |
| } | 076400 | 000175 | Closing (right) brace |
| ~ | 077000 | 000176 | Tilde |
| DEL | 077400 | 000177 | Delete |



APPENDIX B

Summary of Commands



All MPE/3000 commands are summarized below, grouped alphabetically. Following these commands, the MPE/3000 Segmenter commands are also summarized. In both summaries, the last column denotes when the command can be issued: during a batch job (J), during a time-sharing session (S), during a break (B), or programmatically (through the COMMAND intrinsic) (P).

STANDARD CAPABILITY COMMANDS

| Command | Parameters | Function | When Issued | Text Discussion |
|------------|-------------------------------------------------------------------------------|------------------------------------------------------------------|-------------|-----------------|
| :ABORT | | Aborts the current program. | B | 3-23 |
| :ALTSEC | <i>filereference</i> [:({ <i>modelist</i> : <i>userlist</i> [:...]})] | Changes security provisions for a file. | J,S,B,P | 5-51 |
| :BASIC | [<i>commandfile</i>] [, [<i>inputfile</i>] [, [<i>listfile</i>]]] | Calls BASIC/3000 interpreter. | J,S | 4-5 |
| :BASICOMP | [<i>textfile</i>][, [<i>uslfile</i>] [, [<i>listfile</i>]]] | Compiles a SAVE-FAST BASIC/3000 program. | J,S | 4-7 |
| :BASICPREP | [<i>textfile</i>] [, [<i>progfile</i>] [, [<i>listfile</i>]]] | Compiles and prepares a SAVE-FAST BASIC/3000 program. | J,S | 4-9 |
| :BASICGO | [<i>textfile</i>][, <i>listfile</i>] | Compiles, prepares, and executes a SAVE-FAST BASIC/3000 program. | J,S | 4-11 |

| Command | Parameters | Function | When Issued | Text Discussion |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------|-------------|-----------------|
| <i>:BUILD</i> | <i>filereference</i> [;DEV=device] [;DISC=[filesize] [, [numextents] [,initialloc]]] [;REC=[recsize] [, [blockfactor] [[[F] [,BINARY]]]] [[[U] [,ASCII]]]]] [;CCTL] [;TEMP] [;CODE=filecode] | Creates a new file. | J,S,B,P | 5-30 |
| <i>:BYE</i> | | Terminates a session. | S | 3-22 |
| <i>:COBOL</i> | [<i>textfile</i>] [, [<i>usfile</i>] [, [<i>listfile</i>] [, [<i>masterfile</i>] [, [<i>newfile</i>]]]] | Compiles a COBOL/3000 program. | J,S | 4-7 |
| <i>:COBOLGO</i> | [<i>textfile</i>] [, [<i>listfile</i>] [, [<i>masterfile</i>] [, [<i>newfile</i>]]] | Compiles, prepares, and executes a COBOL/3000 program. | J,S | 4-11 |
| <i>:COBOLPREP</i> | [<i>textfile</i>] [, [<i>progfile</i>] [, [<i>listfile</i>] [, [<i>masterfile</i>] [, [<i>newfile</i>]]]] | Compiles and prepares a COBOL/3000 program. | J,S | 4-9 |
| <i>:COMMENT</i> | <i>text</i> | Inserts comments in command stream. | J,S,B,P | 8-6 |
| <i>:CONTINUE</i> | | Disregards job-error condition. | J | 3-25 |
| <i>:DATA</i> | [<i>jobname</i> ,] <i>username</i> [/ <i>upass</i>] <i>.acctname</i> [/ <i>apass</i>] [, <i>filename</i>] | Defines data from outside standard input stream. Cannot be read on \$STDINX file. Acceptable for device recognition. | J,S | 3-24 |
| <i>:EDITOR</i> | [<i>listfile</i>] | Calls the EDITOR. | J,S | 4-22 |
| <i>:EOD</i> | | Denotes end of data. Cannot be read on \$STDINX file. | J,S | 3-16 |
| <i>:EOJ</i> | | Denotes end of batch job. Cannot be read on \$STDINX file. | J | 3-16 |

| Command | Parameters | Function | When Issued | Text Discussion |
|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------|-------------|-----------------|
| <i>:FILE</i> | (This command has many parameters that can be combined in various different formats; the user is referred to Section V.) | Defines or redefines a file's characteristics. | J,S,B,P | 5-11 |
| <i>:FORTGO</i> | <i>[textfile] [, [listfile] [, [masterfile] [, newfile]]]</i> | Compiles, prepares, and executes a FORTRAN/3000 program. | J,S | 4-11 |
| <i>:FORTPREP</i> | <i>[textfile] [, [progfile] [, [listfile] [, [masterfile] [, newfile]]]</i> | Compiles and prepares FORTRAN/3000 program. | J,S | 4-9 |
| <i>:FORTRAN</i> | <i>[textfile] [, [usfile] [, [listfile] [, [masterfile] [, newfile]]]</i> | Compiles a FORTRAN program. | J,S | 4-7 |
| <i>:FREERIN</i> | <i>rin</i> | Deallocates a global RIN, and returns it to RIN pool. | J,S | 9-6 |
| <i>:GETRIN</i> | <i>[rinpassword]</i> | Acquires a global RIN. | J,S,B,P | 9-2 |
| <i>:HELLO</i> | <i>[sessionname,] username [/upass] .acctname [/apass] [, group [/gpass]] [:TERM=termtype] [:TIME=cputime] [:PRI=executionpriority] [:HIPRI [:INPRI=inputpriority]</i> | Initiates a session. Acceptable for device recognition. Requires IA capability class. | S | 3-19 |
| <i>:JOB</i> | <i>[jobname,] username [/upass] .acctname [/apass] [, groupname [/gpass]] [:TERM=termtype] [:TIME=cputime] [:PRI=executionpriority] [:HIPRI [:INPRI=inputpriority] [:RESTART] [:OUTCLASS=device] [, [outputpriority] [, numcopies]]]</i> | Initiates a batch job. Cannot be read on \$STDINX. Acceptable for device recognition. Requires BA capability class. | J,S | 3-11 |
| <i>:LISTF</i> | <i>[fileset] [, detail] [:listfile]</i> | Lists descriptions of files. | J,S,B,P | 5-32 |
| <i>:PREP</i> | <i>usfile, progfile [:ZERODB] [:PMAP] [:MAXDATA=segsz] [:STACK=stacksize]</i> | Prepares a compiled program into segmented form. | J,S | 4-12 |

| Command | Parameters | Function | When Issued | Text Discussion |
|------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------|-------------|-----------------|
| | [;DL=dsize] [;CAP=caplist] [;RL=filename] | | | |
| :PREPRUN | uslfile [,entrypoint] [;NOPRIV] [;PMAP] [;DEBUG] [;LMAP] [;ZERODB] [;MAXDATA=segsz] [;PARM=parameternum] [;STACK=stacksize] [;DL=dsize] [;LIB=library] [;CAP=caplist] [;RL=filename] | Prepares and executes a program. | J,S | 4-17 |
| :PTAPE | [filename] | Reads a paper tape without X-OFF control. | S,B,P | 8-87 |
| :PURGE | filereference [,TEMP] | Deletes a file from the system. | J,S,B,P | 5-32 |
| :RELEASE | filereference | Releases the security provisions of a file. | J,S,B,P | 5-53 |
| :RENAME | oldfilereference newfilereference [,TEMP] | Renames a file. | J,S,B,P | 5-44 |
| :REPORT | | Displays total accounting information for a log-on group. | J,S,B,P | 8-2 |
| :RESET | { @ formaldesignator } | Resets a formal file designator | J,S,B,P | 5-29 |
| :RESETDUMP | | Disables Abort Stack Dump facility. | J,S,B,P | 8-65 |
| :RESTORE | tapefile[;{filessetlist} [;KEEP] [;DEV=device] [;SHOW] | Restores a complete filesset, stored off-line. | J,S,B,P | 5-40 |
| :RESUME | | Resumes an interrupted program. | B | 3-22,8-46 |

| Command | Parameters | Function | When Issued | Text Discussion |
|------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------|-------------|-----------------|
| :RJE | <i>[commandfile][,[inputfile] [,,[listfile][,punchfile]]]</i> | Starts Remote Job Entry subsystem. | J,S | 4-25 |
| :RPG | <i>[textfile][,[uslfile] [,,[listfile][,[masterfile] [,newfile]]]]]</i> | Compiles an RPG/3000 program. | J,S | 4-7 |
| :RPGGO | <i>[textfile][,[listfile] [,,[masterfile][,newfile]]]</i> | Compiles, prepares, and executes an RPG/3000 program. | J,S | 4-11 |
| :RPGPREP | <i>[textfile][,[progfile] [,,[listfile][,[masterfile] [,newfile]]]]]</i> | Compiles and prepares an RPG/3000 program. | J,S | 4-9 |
| :RUN | <i>progfile[,entrypoint] [;NOPRIV] [;LMAP] [;DEBUG] [;MAXDATA=segsize] [;PARM=parameternum] [;STACK=stacksize] [;DL=dsize] [;LIB=library]</i> | Loads and executes a program. | J,S | 4-21 |
| :SAVE | $\left\{ \begin{array}{l} [\$OLDPASS,newfilereference] \\ [tempfilereference] \end{array} \right\}$ | Changes a file to permanent status. Requires SF capability for saving files. | J,S,B,P | 5-31 |
| :SECURE | <i>filereference</i> | Restores suspended security provisions for a file. | J,S,B,P | 5-53 |
| :SEGMENTER | <i>[listfile]</i> | Calls MPE/3000 Segmenter. | J,S | 4-24,7-2 |
| :SETDUMP | <i>[DB[,ST[,QS]]][;ASCII]</i> | Enables stack dump on abort. | J,S,B,P | 8-64 |
| :SETMSG | $\left\{ \begin{array}{l} ON \\ OFF \end{array} \right\}$ | Enables or disables reception of messages. | J,S,B,P | 8-9 |
| :SHOWDEV | $\left[\begin{array}{l} /dev \\ classname \end{array} \right]$ | Displays device information. | J,S,B,P | 8-4 |
| :SHOWIN | $\left[\begin{array}{l} \#Innn \\ STATUS \\ qualifier[,qualifier] . . . \end{array} \right]$ | Displays input devicefile information. | J,S,B,P | 8-5 |

| Command | Parameters | Function | When Issued | Text Discussion |
|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------|-------------|-----------------|
| :SHOWJOB | $\left[\begin{array}{l} \textit{jsnumber} \\ \textit{STATUS} \\ \textit{qualifier 1}[:\textit{qualifier 2}] \\ \textit{qualifier 2}[:\textit{qualifier 1}] \end{array} \right]$ | Displays job/session information. | J,S,B,P | 8-2 |
| :SHOWOUT | $\left[\begin{array}{l} \textit{\#Onnn} \\ \textit{STATUS} \\ \textit{qualifier}[:\textit{qualifier}] \dots \end{array} \right]$ | Displays output devicefile information. | J,S,B,P | 8-5 |
| :SHOWTIME | | Displays current date and time-of-day. | J,S,B,P | 8-2 |
| :SPEED | $\left\{ \begin{array}{l} [\textit{inspeed}],\textit{outspeed} \\ \textit{inspeed} \end{array} \right\}$ | Changes input speed or output speed of terminal. | S,B,P | 8-60 |
| :SPL | $[\textit{sourcefile}][, [\textit{uslfile}][, [\textit{listfile}][, [\textit{masterfile}][, [\textit{newfile}]]]]]$ | Compiles an SPL/3000 program. | J,S | 4-7 |
| :SPLGO | $[\textit{sourcefile}][, [\textit{listfile}][, [\textit{masterfile}][, [\textit{newfile}]]]]]$ | Compiles, prepares, and executes an SPL/3000 program. | J,S | 4-11 |
| :SPLPREP | $[\textit{sourcefile}][, [\textit{progfile}][, [\textit{listfile}][, [\textit{masterfile}][, [\textit{newfile}]]]]]$ | Compiles and prepares an SPL/3000 program. | J,S | 4-9 |
| :STAR | $[\textit{listfile}][, [\textit{NOLIST}]]]$ | Calls the Statistical Analysis Routines. | J,S | 4-23 |
| :STORE | $[\textit{filesetlist}];\textit{tapefile}[:\textit{SHOW}]$ | Stores a set of files off-line. | J,S,B,P | 5-35 |
| :STREAM | $[\textit{filedesignator}][, [\textit{character}]]]$ | Introduces jobs or data decks to MPE/3000. | J,S,B,P | 3-28 |
| :TELL | $\left\{ \begin{array}{l} \textit{jsnumber} \\ \textit{jsname} \end{array} \right\} ;\textit{message}$ | Transmits a message. | J,S,B,P | 8-8 |
| :TELLOP | $\textit{message}$ | Transmits a message from the user to the computer operator. | J,S,B,P | 8-9 |

STANDARD CAPABILITY COMMANDS (SEGMENTER COMMANDS)

| Command | Parameters | Function | When Issued | Text Discussion |
|-----------|------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------|-------------|-----------------|
| -ADDRL | <i>name[(index)]</i> | Adds a procedure to an RL. | J,S | 7-19 |
| -ADDSL | <i>name[,PMAP]</i> | Adds a segment to an SL. | J,S | 7-24 |
| -AUXUSL | <i>filereference</i> | Designates a source of RBM input for -COPY command. | J,S | 7-9 |
| -BUILDRL | <i>filereference,records,extents</i> | Creates a permanent, formatted RL file. | J,S | 7-18 |
| -BUILDSL | <i>filereference,records,extents</i> | Creates a permanent, formatted SL file. | J,S | 7-23 |
| -BUILDUSL | <i>filereference,records,extents</i> | Creates a temporary, formatted USL file. | J,S | 7-3 |
| -CEASE | <i>[pointspec,] name[(index)]</i> | Deactivates one or more entry-points in a USL. | J,S | 7-6 |
| -COPY | <i>[rbmspec,] name[(index)]</i> | Copies an RBM or segment from one USL to another. | J,S | 7-9 |
| -EXIT | | Exits from Segmenter, returning control to MPE/3000 Command Interpreter. | J,S | 7-2 |
| -HIDE | <i>name[(index)]</i> | Sets an RBM internal flag <i>on</i> . | J,S | 7-27 |
| -LISTRL | | Lists the procedures in an RL. | J,S | 7-20 |
| -LISTSL | | Lists the segments in an SL. | J,S | 7-24 |
| -LISTUSL | | Lists the RBM's in a USL. | J,S | 7-10 |
| -NEWSEG | <i>newsegname,rbmname [(index)]</i> | Changes the segment name of an RBM. | J,S | 7-8 |
| -PREPARE | <i>progfile [:ZERODB] [:PMAP] [:MAXDATA=segsz] [:STACK=stacksz] [:DL=dlsz] [:CAP=caplist] [:RL=filename]</i> | Prepares RBM's from a USL into a program file. | J,S | 7-12 |
| -PURGERBM | <i>[rbmspec,] name[(index)]</i> | Deletes one or more RBM's from a USL. | J,S | 7-7 |

| Command | Parameters | Function | When Issued | Text Discussion |
|----------|-----------------------------------|--------------------------------------------------|-------------|-----------------|
| -PURGERL | <i>[rbmspec,] name</i> | Deletes an entrypoint or a procedure from an RL. | J,S | 7-19 |
| -PURGESL | <i>[unitspec,] name</i> | Deletes an entrypoint or a segment from an SL. | J,S | 7-24 |
| -REVEAL | <i>name[(index)]</i> | Sets an RBM internal flag <i>off</i> . | J,S | 7-27 |
| -RL | <i>filereference</i> | Designates an RL for management. | J,S | 7-19 |
| -SL | <i>filereference</i> | Designates an SL for management. | J,S | 7-23 |
| -USE | <i>[pointspec,] name[(index)]</i> | Activates one or more RBM entry-points. | J,S | 7-5 |
| -USL | <i>filereference</i> | Designates a USL for management. | J,S | 7-3 |

BREAKABLE COMMANDS

All MPE user commands fall into one of two major categories: breakable and non-breakable. Non-breakable commands always complete; striking the break key has no effect on their execution. Breakable commands are either placed in a suspended state or aborted, depending on the type of command. Table B-1 categorizes all user command according to type.

The following rules apply:

1. A suspended command may be resumed or aborted by issuing the :RESUME or :ABORT command.

In general, any non-suspendable command may be issued in suspended mode, with the following exceptions:

```

:BYE           :CONTINUE       :ALLOCATE
:DATE          :FREERIN
:EOD           :HELLO
:EOJ           :JOB

```

2. Abortable commands are aborted upon detecting a break. This rule holds even when the command is being executed in suspended mode.
3. All user commands are considered non-abortable when being executed via the COMMAND intrinsic. In this case they are suspendable.

4. For those abortable commands capable of having their printed output directed to an alternate file, this output will, when broken, prematurely terminate in the same manner as if it were directed to the job list device.
5. Striking the break key during the execution of a non-breakable command has no effect upon command execution.

When a break is pressed for a suspendable command, control is returned to the MPE command interpreter. At this point the user can:

- a. Issue "minor" system commands to send messages, examine status, create/delete files, etc. Commands which require an abortion of the current program will print:

ABORT? $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$

before executing the command. If the user enters "NO", then the command is not executed and the program is not aborted; "YES" will cause an abnormal termination of the program and the command will be executed.

- b. The program can be abnormally terminated by entering:

:ABORT

- c. The program can be resumed, restoring it to its pre-broken state, with:

:RESUME

If the program had a read pending on the terminal when break was passed, the message "READ PENDING" is issued and the user must satisfy this pending read; all characters entered before break are retained.

The :ABORT and :RESUME commands are legitimate only during the suspended state.

Table B-1. Breakable and Non-Breakable Commands

| NON-BREAKABLE | | BREAKABLE | |
|---------------|-----------|------------|-------------|
| | | SUSPENDED | ABORTED |
| :ABORT | :ALTSEC | :BASIC | :LISTF |
| :BUILD | :BYE | :COBOL | :HELLO * |
| :COMMENT | :CONTINUE | :COBOLGO | :REPORT |
| :DATA | :EOD | :COBOLPREP | :STORE ** |
| :EOJ | :FILE | :RPG | :RESTORE ** |
| :FREERIN | :GETRIN | :RPGGO | :SHOWJOB |
| :JOB | :PTAPE | :RPGPREP | :SHOWDEV |
| :PURGE | :RELEASE | :RJE | :STREAM |
| :RENAME | :RESET | :BASICOMP | :SHOWIN |
| :RESUME | :SAVE | :BASICGO | :SHOWOUT |
| :SECURE | :SHOWTIME | :BASICPREP | |
| :SPEED | :TELL | :EDITOR | |
| :TELLOP | :SETDUMP | :FORTRAN | |
| :RESETDUMP | :SETMSG | :FORTGO | |
| | | :FORTPREP | |
| | | :PREP | |
| | | :PREPRUN | |
| | | :RUN | |
| | | :SEGMENTER | |
| | | :SPL | |
| | | :SPLGO | |
| | | :SPLPREP | |
| | | :STAR | |

* Breaking :HELLO will suppress the welcome message, if any exists.
 ** :STORE/:RESTORE, when broken, will stop after completing the current file and will suppress further output.

APPENDIX C

Summary of Intrinsic Calls

All intrinsic calls available to the user are summarized below, listed alphabetically. For each intrinsic, the complete declaration head appears; the intrinsic call format is distinguished from the remainder of the head by a box. The function of the intrinsic is described. For those intrinsics that are type procedures, the procedure *type* is noted. Optional parameters are **bold face** in the intrinsic head and are also noted separately. All condition codes that can be returned by the intrinsic are listed. The intrinsic error-code number is also presented. (This is the number that appears in the abort-error message generated when an error is encountered in the corresponding intrinsic.) The functional categories represented by the error number set are

| Error Number Range | Functional Category |
|--------------------|------------------------|
| 1 – 29 | File Management |
| 30 – 39 | Resource Management |
| 40 – 49 | System Timer (Clock) |
| 50 – 59 | Traps |
| 60 – 79 | Utility Routines |
| 80 – 99 | Program Management |
| 100 – 119 | Process Control |
| 120 – 129 | Scheduling |
| 130 – 149 | Data Segments |
| 180 – 199 | Input/Output Utilities |
| 200 – 209 | Special Utilities |

Where intrinsics have special attributes, those attributes are noted as follows:

- Uncallable intrinsics (those that can only be invoked by the user in privileged mode) are indicated by an asterisk (*).
- Intrinsics that can be called in privileged mode even though the user does not have the appropriate capability are denoted by a plus sign (+).
- Intrinsics that can be called by a process when the DB register is not pointing to that process' stack are denoted by a dollar sign (\$).

The MPE/3000 Optional Capability (if any) required to invoke the intrinsic, and the page in the main text where the intrinsic is discussed, are also noted.

PROCEDURE

ACTIVATE (pin, susp);

VALUE *pin, susp;*

LOGICAL *susp;*

INTEGER *pin;*

OPTION VARIABLE, EXTERNAL;

FUNCTION: *Activates a process.*

TYPE: *None*

OPTIONAL PARAMETERS: *susp*

CONDITION CODES: *CCE, CCG, CCL*

ERROR CODE: *104*

SPECIAL ATTRIBUTES: *\$*

OPTIONAL CAPABILITY: *Process Handling*

TEXT DISCUSSION: *11-12*

INTEGER PROCEDURE

ADJUSTUSLF (uslfnm, records);

VALUE uslfnm, records;

INTEGER uslfnm; records;

OPTION EXTERNAL;

FUNCTION: Moves information block on a USL file.

TYPE: Integer

OPTIONAL PARAMETERS: None

CONDITION CODES: CCE, CCL

ERROR CODE: 83

SPECIAL ATTRIBUTES: None

OPTIONAL CAPABILITY: None

TEXT DISCUSSION: 8-72

PROCEDURE

ALTDSEG (index, inc, size);

VALUE *index, size;*

LOGICAL *index;*

INTEGER *inc, size;*

OPTIONAL EXTERNAL;

FUNCTION: *Changes size of an extra data segment.*

TYPE: *None*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *CCE, CCG, CCL*

ERROR CODE: *134*

SPECIAL ATTRIBUTE *+*

OPTIONAL CAPABILITY: *Data Segment Management*

TEXT DISCUSSION: *12-7*

PROCEDURE

ARITRAP (state);

VALUE state;

LOGICAL state;

OPTION EXTERNAL;

FUNCTION: *Enables or disables internal interrupt signals from all hardware arithmetic traps.*

TYPE: *None*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *CCE, CCG*

ERROR CODE: *51*

SPECIAL ATTRIBUTES: *None*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *8-31*

INTEGER PROCEDURE

ASCII (word, base, string);

VALUE *word, base;*

LOGICAL *word;*

INTEGER *base;*

BYTE ARRAY *string;*

OPTION EXTERNAL;

FUNCTION: *Converts a number from binary to ASCII code.*

TYPE: *Integer*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *None*

ERROR CODE: *63*

SPECIAL ATTRIBUTES: *None*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *8-11*

LOGICAL PROCEDURE

BINARY (string, length);

VALUE length;

BYTE ARRAY string;

INTEGER length;

OPTION EXTERNAL;



FUNCTION: *Converts a number from ASCII to binary code.*

TYPE: *Logical*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *CCE, CCG, CCL*

ERROR CODE: *62*

SPECIAL ATTRIBUTES: *None*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *8-10*

PROCEDURE

CAUSEBREAK;

OPTION EXTERNAL;

FUNCTION: *Requests a session break.*

TYPE: *None*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *CCE, CCL*

ERROR CODE: *56*

SPECIAL ATTRIBUTES: *\$*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *8-46*

LOGICAL PROCEDURE

CHECKDEV (ldev, hdevaddr);

VALUE *ldev;*

INTEGER *ldev, hdevaddr;*

OPTION EXTERNAL;

FUNCTION: *Verifies input/output device acquisition*

TYPE: *Logical*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *CCE, CCG*

ERROR CODE: *None*

SPECIAL ATTRIBUTES: *None*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *8-70*

LOGICAL PROCEDURE
OPTION EXTERNAL;

CALENDAR;

| | |
|-----------------------------|-----------------------------|
| <i>FUNCTION:</i> | <i>Returns current date</i> |
| <i>TYPE:</i> | <i>Logical</i> |
| <i>OPTIONAL PARAMETERS:</i> | <i>None</i> |
| <i>CONDITION CODES:</i> | <i>None</i> |
| <i>ERROR CODE:</i> | <i>None</i> |
| <i>SPECIAL ATTRIBUTES:</i> | <i>\$</i> |
| <i>OPTIONAL CAPABILITY:</i> | <i>None</i> |
| <i>TEXT DISCUSSION:</i> | <i>8-18</i> |

LONG PROCEDURE

CHRONOS;

OPTION EXTERNAL;

FUNCTION: *Returns current date and time.*

TYPE: *Long*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *None*

ERROR CODE: *41*

SPECIAL ATTRIBUTES: *\$*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *8-18*

DOUBLE PROCEDURE

CLOCK;

OPTION EXTERNAL;

FUNCTION: Returns current time

TYPE: Double

OPTIONAL PARAMETERS: None

CONDITION CODES: None

ERROR CODE: None

SPECIAL ATTRIBUTES: \$

OPTIONAL CAPABILITY: None

TEXT DISCUSSION: 8-18

PROCEDURE

COMMAND (comimage, error, parm);

BYTE ARRAY comimage;

INTEGER error, parm;

OPTIONAL EXTERNAL;

FUNCTION: *Executes an MPE/3000 command programmatically.*

TYPE: *None*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *CCE, CCG, CCL*

ERROR CODE: *68*

SPECIAL ATTRIBUTES: *None*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *8-29*

PROCEDURE

| |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>CREATE</i> (<i>progrname</i> , <i>entryname</i> , <i>pin</i> , <i>param</i> , <i>flags</i> , <i>stacksize</i> , <i>dlsiz</i> e, <i>maxdata</i> , <i>priorityclass</i> , <i>rank</i>); |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

VALUE *param*, *stacksize*, *dlsiz*e, *priority-class*, *maxdata*, *flags*, *rank*;

LOGICAL *priorityclass*, *flags*;

INTEGER *stacksize*, *dlsiz*e, *maxdata*, *pin*, *param*, *rank*;

BYTE ARRAY *progrname*, *entryname*;

OPTION VARIABLE, EXTERNAL;

FUNCTION: *Creates a process.*

TYPE: *None*

OPTIONAL PARAMETERS: *entryname*, *param*, *flags*, *stacksize*, *dlsiz*e, *maxdata*,
priorityclass, *rank*

CONDITION CODES: *CCE, CCG, CCL*

ERROR CODE: *100*

SPECIAL ATTRIBUTES: *None*

OPTIONAL CAPABILITY: *Process Handling*

TEXT DISCUSSION: *11-7*

PROCEDURE *CTRANSLATE (code, instring, outstring, stringlength, table);*

VALUE *code, stringlength;*

INTEGER *code, stringlength;*

BYTE ARRAY *instring, outstring, table;*

OPTION VARIABLE, EXTERNAL;

FUNCTION: *Performs character translation*

TYPE: *None*

OPTIONAL PARAMETERS: *outstring*

CONDITION CODES: *CCE, CCL*

ERROR CODE: *202*

SPECIAL ATTRIBUTES: *None*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *8-13*



INTEGER PROCEDURE

DASCII (dword, base, string);

VALUE *dword, base;*

DOUBLE *dword;*

INTEGER *base;*

BYTE ARRAY *string;*

OPTION EXTERNAL;

FUNCTION: *Converts a value from double-word binary to ASCII.*

TYPE: *Integer*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *None*

ERROR CODE: *75*

SPECIAL ATTRIBUTES: *None*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *8-12*

DOUBLE PROCEDURE

DBINARY (string, length);

VALUE *length;*

BYTE ARRAY *string;*

INTEGER *length;*

OPTION EXTERNAL;

FUNCTION: *Converts a number from ASCII to double-word binary value.*

TYPE: *Double*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *CCE, CCG, CCL*

ERROR CODE: *74*

SPECIAL ATTRIBUTES: *None*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *8-11*

PROCEDURE

DEBUG

OPTION EXTERNAL;

| | |
|-----------------------------|------------------------------------------------------------------------------|
| <i>FUNCTION:</i> | <i>Sets breakpoints and modifies or displays stack or register contents.</i> |
| <i>TYPE:</i> | <i>None</i> |
| <i>OPTIONAL PARAMETERS:</i> | <i>None</i> |
| <i>CONDITION CODES:</i> | <i>None</i> |
| <i>ERROR CODE:</i> | <i>99</i> |
| <i>SPECIAL ATTRIBUTES:</i> | <i>None</i> |
| <i>OPTIONAL CAPABILITY:</i> | <i>None</i> |
| <i>TEXT DISCUSSION:</i> | <i>8-50</i> |

INTEGER PROCEDURE

DLSIZE (size);

VALUE *size;*

INTEGER *size;*

OPTION EXTERNAL;

FUNCTION: *Changes size of DL-DB area.*

TYPE: *Integer*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *CCE, CCG, CCL*

ERROR CODE: *135*

SPECIAL ATTRIBUTES: *None*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *8-44*

PROCEDURE

DMOVIN (index, disp, number, location);

VALUE *index, disp, number;*

LOGICAL *index;*

INTEGER *disp, number;*

ARRAY *location;*

OPTION EXTERNAL;

FUNCTION: *Copies block from data segment to stack.*

TYPE: *None*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *CCE, CCG, CCL*

ERROR CODE: *132*

SPECIAL ATTRIBUTES: *+*

OPTIONAL CAPABILITY: *Data Segment Management*

TEXT DISCUSSION: *12-4*

PROCEDURE

DMOVOUT (index, disp, number, location);

VALUE *index, disp, number;*

LOGICAL *index;*

INTEGER *disp, number;*

ARRAY *location;*

OPTION EXTERNAL;

FUNCTION: *Copies block from stack to data segment.*

TYPE: *None*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *CCE, CCG, CCL*

ERROR CODE: *133*

SPECIAL ATTRIBUTES: *+*

OPTIONAL CAPABILITY: *Data Segment Management*

TEXT DISCUSSION: *12-6*

INTEGER PROCEDURE

FATHER;

OPTION EXTERNAL;

FUNCTION: *Requests PIN of father process.*

TYPE: *Integer*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *CCE, CCG, CCL*

ERROR CODE: *109*

SPECIAL ATTRIBUTES: *\$*

OPTIONAL CAPABILITY: *Process Handling*

TEXT DISCUSSION: *11-25*

PROCEDURE

FCHECK (*filenum, errorcode, tlog, blknum, numrecs*);

VALUE *filenum*;

INTEGER *filenum, errorcode, tlog, numrecs*;

DOUBLE *blknum*;

OPTION VARIABLE; EXTERNAL;

FUNCTION: *Requests details about file input/output errors.*

TYPE: *None*

OPTIONAL PARAMETERS: *errorcode, tlog, blknum, numrecs*

CONDITION CODES: *CCE, CCL*

ERROR CODE: *10*

SPECIAL ATTRIBUTES: *\$*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *6-52*

PROCEDURE

FCLOSE (filenum, disposition, seccode);

VALUE *filenum, disposition, seccode;*

INTEGER *filenum, disposition, seccode;*

OPTION EXTERNAL;

FUNCTION: *Closes a file.*

TYPE: *None*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *CCE, CCL*

ERROR CODE: *9*

SPECIAL ATTRIBUTES: *\$*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *6-30*

PROCEDURE

FCONTROL (filenum, controlcode, param);

VALUE *filenum, controlcode;*

INTEGER *filenum, controlcode;*

LOGICAL *param;*

OPTION EXTERNAL;

FUNCTION: *Performs various control operations on a file or terminal device.*

TYPE: *None*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *CCE, CCL, CCG*

ERROR CODE: *13*

SPECIAL ATTRIBUTES: *\$*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *6-56, 8-80, 8-84, 8-85, 8-86, 8-88, 8-89, 8-90*

PROCEDURE

FGETINFO (*filenum*, *filename*, *foptions*, *aoptions*, *reclsize*,
devtype, *ldnum*, *hdaddr*, *filecode*, *recpt*, *eof*,
flimit, *logcount*, *physcount*, *blksize*, *extsize*,
numextents, *userlabels*, *creatorid*, *labaddr*);

VALUE: *filenum*;

INTEGER: *filenum*, *reclsize*, *detype*, *filecode*, *blksize*, *numextents*, *userlabels*;

BYTE ARRAY *filename*, *creatorid*;

LOGICAL *foptions*, *aoptions*, *ldnum*, *hdaddr*, *extsize*;

DOUBLE *recpt*, *eof*, *flimit*, *logcount*, *physcount*, *labaddr*;

OPTION VARIABLE, EXTERNAL;

FUNCTION: *Requests access and status information about a file.*

TYPE: *None*

OPTIONAL PARAMETERS: *filename*, *foptions*, *aoptions*, *reclsize*, *detype*, *ldnum*, *hdaddr*,
filecode, *recpt*, *eof*, *flimit*, *logcount*, *physcount*, *blksize*, *extsize*,
numextents, *userlabels*, *creatorid*, *labaddr*

CONDITION CODES: *CCE*, *CCL*

ERROR CODE: *11*

SPECIAL ATTRIBUTES: *None*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *6-48*

PROCEDURE

FLOCK (filenum, lockcond);

VALUE *filenum, lockcond;*

INTEGER *filenum;*

LOGICAL *lockcond;*

OPTION EXTERNAL;

FUNCTION: *Dynamically locks a file.*

TYPE: *None*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *CCE, CCG, CCL*

ERROR CODE: *15*

SPECIAL ATTRIBUTES: *\$*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *9-10*

INTEGER PROCEDURE

| |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>FOPEN</i> (formaldesignator, foptions, aoptions, resize, device, formmsg, userlabels, blockfactor, numbuffers, filesize, numextents, initialloc, filecode); |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------|

VALUE foptions, aoptions, resize, userlabels, blockfactor, numbuffers, filesize, numextents, initialloc, filecode;

BYTE ARRAY formaldesignator, device, formmsg;

LOGICAL foptions, aoptions;

INTEGER resize, userlabels, blockfactor, numbuffers, numextents, initialloc, filecode;

DOUBLE filesize;

OPTION VARIABLE, EXTERNAL;

FUNCTION: Opens a file.

TYPE: Integer

OPTIONAL PARAMETERS: formaldesignator, foptions, aoptions, resize, device, formmsg, userlabels, blockfactor, numbuffers, filesize, numextents, initialloc, filecode

CONDITION CODES: CCE, CCL

ERROR CODE: 1

SPECIAL ATTRIBUTES: None

OPTIONAL CAPABILITY: None

TEXT DISCUSSION: 6-17

PROCEDURE

FPOINT (filenum, recnum);

VALUE *filenum, recnum;*

INTEGER *filenum;*

DOUBLE *recnum;*

OPTION EXTERNAL;

FUNCTION: *Resets the logical record pointer for a sequential disc file.*

TYPE: *None*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *CCE, CCG, CCL*

ERROR CODE: *6*

SPECIAL ATTRIBUTES: *\$*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *6-47*

INTEGER PROCEDURE

FREAD (filenum, target, tcount);

VALUE *filenum, tcount;*

INTEGER *filenum, tcount;*

ARRAY *target;*

OPTION EXTERNAL;

FUNCTION: *Reads a logical record from a sequential file (on any device) to the user's stack.*

TYPE: *Integer*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *CCE, CCG, CCL*

ERROR CODE: *2*

SPECIAL ATTRIBUTES: *\$*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *6-32*

PROCEDURE

FREADDIR (filenum, target, tcount, recnum);

VALUE *filenum, tcount, recnum;*

INTEGER *filenum;*

ARRAY *target;*

INTEGER *tcount;*

DOUBLE *recnum;*

OPTION EXTERNAL;

FUNCTION: *Reads a logical record from a direct-access disc file to the user's data stack.*

TYPE: *None*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *CCE, CCG, CCL*

ERROR CODE: *7*

SPECIAL ATTRIBUTES: *\$*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *6-34*

PROCEDURE

FREADLABEL (filenum, target, tcount); labelid

VALUE *filenum, tcount, labelid*

INTEGER *filenum, tcount, labelid*

ARRAY *target*

OPTION VARIABLE, EXTERNAL;

FUNCTION: *Reads a user file label.*

TYPE: *None*

OPTIONAL PARAMETERS: *tcount*

CONDITION CODES: *CCE, CCG, CCL*

ERROR CODES: *19*

SPECIAL ATTRIBUTES: *\$*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *6-43*

PROCEDURE

FREADSEEK (filenum, recnum);

VALUE *filenum, recnum;*

INTEGER *filenum;*

DOUBLE *recnum;*

OPTION EXTERNAL;

FUNCTION: *Prepares, in advance, for reading from a direct-access file.*

TYPE: *None*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *CCE, CCG, CCL*

ERROR CODE: *12*

SPECIAL ATTRIBUTES: *\$*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *6-36*

PROCEDURE

FREEDSEG (index, id);

VALUE *index, id;*

LOGICAL *index, id;*

OPTIONAL EXTERNAL;

FUNCTION: *Releases an extra data segment.*

TYPE: *None*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *CCE, CCG, CCL*

ERROR CODE: *131*

SPECIAL ATTRIBUTES: *+*

OPTIONAL CAPABILITY: *Data Segment Management*

TEXT DISCUSSION: *12-3*

PROCEDURE

FREELOCRIN;

OPTION EXTERNAL;

FUNCTION: *Frees all local RIN's from allocation to a job.*

TYPE: *None*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *CCE, CCG, CCL*

ERROR CODE: *31*

SPECIAL ATTRIBUTES: *None*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *9-9*

LOGICAL PROCEDURE:

FRELATE (infilenum, listfilenum);

VALUE *infilenum, listfilenum;*

INTEGER *infilenum, listfilenum;*

OPTION EXTERNAL;

FUNCTION: *Declares a file pair interactive or duplicative.*

TYPE: *Logical*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *CCE, CCL*

ERROR CODE: *18*

SPECIAL ATTRIBUTES: *\$*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *6-61*

PROCEDURE

FRENAME (filename, newfilereference);

VALUE filename;

INTEGER filename;

BYTE ARRAY newfilereference;

OPTION EXTERNAL;

FUNCTION: Renames a disc file.

TYPE: None

OPTIONAL PARAMETERS: None

CONDITION CODES: CCE, CCL

ERROR CODE: 17

SPECIAL ATTRIBUTES: None

OPTIONAL CAPABILITY: None

TEXT DISCUSSION: 6-60

PROCEDURE

FSETMODE (filenum, modeflags);

VALUE *filenum, modeflags;*

INTEGER *filenum;*

LOGICAL *modeflags;*

OPTION EXTERNAL;

FUNCTION: *Activates or deactivates file-access modes.*

TYPE: *None*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *CCE, CCL*

ERROR CODE: *14*

SPECIAL ATTRIBUTES: *\$*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *6-58*

PROCEDURE

FSPACE (filenum, displacement);

VALUE *filenum, displacement;*

INTEGER *filenum, displacement;*

OPTION EXTERNAL;

FUNCTION: *Spaces forward or backward on a file.*

TYPE: *None*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *CCE, CCG, CCL*

ERROR CODE: *5*

SPECIAL ATTRIBUTES: *\$*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *6-46*

PROCEDURE

FUNLOCK (filenum);

VALUE *filenum;*

INTEGER *filenum;*

OPTION EXTERNAL;

FUNCTION: *Dynamically unlocks a file.*

TYPE: *None*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *CCE, CCG, CCL*

ERROR CODE: *16*

SPECIAL ATTRIBUTES: *\$*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *9-11*

PROCEDURE

FUPDATE (filenum, target, tcount);

VALUE *filenum, tcount;*

INTEGER *filenum, tcount;*

ARRAY *target;*

OPTIONAL EXTERNAL;

FUNCTION: *Updates a logical record residing in a disc file.*

TYPE: *None*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *CCE, CCG, CCL*

ERROR CODE: *4*

SPECIAL ATTRIBUTES: *\$*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *6-45*

PROCEDURE

FWRITE (filenum, target, tcount, control);

VALUE *filenum, tcount, control;*

INTEGER *filenum, tcount;*

ARRAY *target;*

LOGICAL *control;*

OPTION EXTERNAL;

FUNCTION: *Writes a logical record from the user's stack to a sequential file (on any device).*

TYPE: *None*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *CCE, CCG, CCL*

ERROR CODE: *3*

SPECIAL ATTRIBUTES: *\$*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *6-37*

PROCEDURE

FWRITEDIR (filenum, target, tcount, recnum);

VALUE *filenum, tcount, recnum;*

INTEGER *filenum, tcount;*

ARRAY *target;*

INTEGER *tcount;*

DOUBLE *recnum;*

OPTION EXTERNAL;

FUNCTION: *Writes a logical record from the user's stack to a direct-access disc file.*

TYPE: *None*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *CCE, CCG, CCL*

ERROR CODE: *8*

SPECIAL ATTRIBUTES: *\$*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *6-41*

PROCEDURE

FWRITELABEL (filenumber, target, tcount);

VALUE *filenum, tcount;*

INTEGER *filenum, tcount;*

ARRAY *target;*

OPTION VARIABLE, EXTERNAL;

FUNCTION: *Writes a user's file label.*

TYPE: *None*

OPTIONAL PARAMETERS: *tcount*

CONDITION CODES: *CCE, CCG, CCL*

ERROR CODE: *20*

SPECIAL ATTRIBUTES: *\$*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *6-44*

PROCEDURE

GETDSEG (index, length, id);

VALUE *id;*

LOGICAL *index, id;*

INTEGER *length;*

OPTION EXTERNAL;

FUNCTION: *Creates an extra data segment.*

TYPE: *None*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *CCE, CCG, CCL*

ERROR CODE *130*

SPECIAL ATTRIBUTES: *+*

OPTIONAL CAPABILITY: *Data Segment Management*

TEXT DISCUSSION: *12-2*

LOGICAL PROCEDURE

GETJCW;

OPTION EXTERNAL;

FUNCTION: *Fetches contents of job control word.*

TYPE: *Logical*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *None*

ERROR CODE: *73*

SPECIAL ATTRIBUTES: *None*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *8-58*

PROCEDURE

GETLOCRIN (rincount);

VALUE rincount;

LOGICAL rincount;

OPTION EXTERNAL;

FUNCTION: Acquires local RIN's. (The number acquired = rincount.)

TYPE: None

OPTIONAL PARAMETERS: None

CONDITION CODES: CCE, CCG, CCL

ERROR CODE: 30

SPECIAL ATTRIBUTES: None

OPTIONAL CAPABILITY: None

TEXT DISCUSSION: 9-6

INTEGER PROCEDURE

GETORIGIN;

OPTION EXTERNAL;

FUNCTION: *Determines source of activation call.*

TYPE: *Integer*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *None*

ERRCR CODE: *105*

SPECIAL ATTRIBUTES: *\$*

OPTIONAL CAPABILITY: *Process Handling*

TEXT DISCUSSION: *11-25*

PROCEDURE

GETPRIORITY (pin, priorityclass, rank);

VALUE *pin, priorityclass, rank;*

LOGICAL *priorityclass;*

INTEGER *pin, rank;*

OPTION VARIABLE, EXTERNAL;

FUNCTION: *Reschedules a process.*

TYPE: *None*

OPTIONAL PARAMETERS: **rank**

CONDITION CODES: *CCE, CCL*

ERROR CODE: *120*

SPECIAL ATTRIBUTES: *\$*

OPTIONAL CAPABILITY: *Process Handling*

TEXT DISCUSSION: *11-22*

PRIVILEGED-MODE OPTIONAL CAPABILITY INTRINSICS

PROCEDURE

`GETPRIVMODE;`

OPTION EXTERNAL;

FUNCTION: *Dynamically enters privileged mode.*

TYPE: *None*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *CCE, CCG*

ERROR CODE: *200*

SPECIAL ATTRIBUTES: *None*

OPTIONAL CAPABILITY: *Privileged Mode*

TEXT DISCUSSION: *14-3*

INTEGER PROCEDURE

GETPROCID (son);

VALUE son;

LOGICAL son;

OPTION EXTERNAL;

FUNCTION: Requests PIN of a son process.

TYPE: Integer

OPTIONAL PARAMETERS: None

CONDITION CODES: None

ERROR CODE: 112

SPECIAL ATTRIBUTES: None

OPTIONAL CAPABILITY: Process Handling

TEXT DISCUSSION: 11-26

DOUBLE PROCEDURE

GETPROCINFO (pin);

VALUE pin;

INTEGER pin;

OPTION EXTERNAL;

FUNCTION: Requests status information about a father or son process.

TYPE: Double

OPTIONAL PARAMETERS: None

CONDITION CODES: CCE, CCG, CCL

ERROR CODE: 110

SPECIAL ATTRIBUTES: \$

OPTIONAL CAPABILITY: Process Handling

TEXT DISCUSSION: 11-27

PROCEDURE

GETUSERMODE;

OPTION EXTERNAL;

FUNCTION: *Dynamically returns to non-privileged mode.*

TYPE: *None*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *CCE, CCG*

ERROR CODE: *201*

SPECIAL ATTRIBUTES: *None*

OPTIONAL CAPABILITY: *Privileged Mode*

TEXT DISCUSSED: *14-4*

INTEGER PROCEDURE

INITUSLF (uslfnm, rec0);

VALUE *uslfnm;*

INTEGER *uslfnm;*

INTEGER ARRAY *rec 0;*

OPTION EXTERNAL;

FUNCTION: *Initializes buffer for a USL file to the empty state.*

TYPE: *Integer*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *CCE, CCL*

ERROR CODE: *82*

SPECIAL ATTRIBUTES: *None*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *8-71*

PROCEDURE

KILL (pin);

VALUE *pin;*

INTEGER *pin;*

OPTION EXTERNAL;



FUNCTION: *Deletes a process.*

TYPE: *None*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *CCE, CCG, CCL*

ERROR CODE: *102*

SPECIAL ATTRIBUTES: *None*

OPTIONAL CAPABILITY: *Process Handling*

TEXT DISCUSSION: *11-14*

INTEGER PROCEDURE

LOADPROC (procname, lib, plabel);

VALUE lib;

BYTE ARRAY procname;

INTEGER lib, plabel.

OPTION EXTERNAL;

FUNCTION: Dynamically loads a library procedure.

TYPE: Integer

OPTIONAL PARAMETERS: None

CONDITION CODES: CCE, CCL

ERROR CODE: 80

SPECIAL ATTRIBUTES: None

OPTIONAL CAPABILITY: None

TEXT DISCUSSION: 7-28

PROCEDURE

LOCKGLORIN (rinum, lockcond, rinpassword);

VALUE rinum;

LOGICAL rinum, lockcond;

BYTE ARRAY rinpassword;

OPTION EXTERNAL;

FUNCTION: Locks a global RIN.

TYPE: None

OPTIONAL PARAMETERS: None

CONDITION CODES: CCE, CCG

ERROR CODE: 34

SPECIAL ATTRIBUTES: None

OPTIONAL CAPABILITY: None

TEXT DISCUSSION: 9-3

PROCEDURE

LOCKLOCIN (rinnum,lockcond);

VALUE *rinnum;*

LOGICAL *rinnum, lockcond;*

OPTION EXTERNAL;

FUNCTION: *Locks a local RIN.*

TYPE: *None*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *CCE, CCG, CCL*

ERROR CODE: *32*

SPECIAL ATTRIBUTES: *None*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *9-7*

LOGICAL PROCEDURE

MAIL (pin, count);

VALUE *pin;*

INTEGER *pin, count;*

OPTION EXTERNAL;

FUNCTION: *Tests mailbox status.*

TYPE: *Logical*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *CCE, CCG*

ERROR CODE: *106*

SPECIAL ATTRIBUTES: *None*

OPTIONAL CAPABILITY: *Process Handling*

TEXT DISCUSSION: *11-17*

INTEGER PROCEDURE

| |
|------------------------------------------------------------------------------------------|
| <i>MYCOMMAND</i> (<i>comimage, delimiters, maxparms, numparms, parms, dict, defn</i>); |
|------------------------------------------------------------------------------------------|

VALUE *maxparms*;

BYTE ARRAY *comimage, delimiters, dict*;

INTEGER *maxparms, numparms*;

DOUBLE ARRAY *parms*;

BYTE POINTER *defn*;

OPTION VARIABLE, EXTERNAL;

FUNCTION: *Parses (delineates and defines parameters) for user-supplied command image.*

TYPE: *Integer*

OPTIONAL PARAMETERS: *dict, defn*

CONDITION CODES: *CCE, CCG, CCL*

ERROR CODE: *71*

SPECIAL ATTRIBUTES: *None*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *8-25*

PROCEDURE

PAUSE (*internal*);

REAL *internal*;

OPTION EXTERNAL;

FUNCTION: Suspends the calling process for a specified number of seconds.

TYPE: None

OPTIONAL PARAMETERS: None

CONDITION CODES: CCE, CCG, CCL

ERROR CODE: 45

SPECIAL ATTRIBUTES: None

OPTIONAL CAPABILITIES: None

TEXT DISCUSSION: 8-18A



PROCEDURE

PRINT (message, length, control);

VALUE *length, control;*

ARRAY *message;*

INTEGER *length, control;*

OPTION EXTERNAL;

FUNCTION: *Prints character string on job/session listing device.*

TYPE: *None*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *CCE, CCL, CCG*

ERROR CODE: *65*

SPECIAL ATTRIBUTES: *None*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *8-15*

PROCEDURE

PRINTOP (message, length, control);

VALUE length, control;

ARRAY message;

INTEGER length, control;

OPTION EXTERNAL;

FUNCTION: Prints a character string on Operator's Console.

TYPE: None

OPTIONAL PARAMETERS: None

CONDITION CODES: CCE, CCL, CCG

ERROR CODE: 66

SPECIAL ATTRIBUTES: None

OPTIONAL CAPABILITY: None

TEXT DISCUSSION: 8-16

INTEGER PROCEDURE

PRINTOPREPLY (message, length, control, reply, expectedl);

VALUE length, control, expectedl;

ARRAY message, reply;

INTEGER length, control, expectedl;

OPTION EXTERNAL;

FUNCTION: Prints a character string on Operator's Console and solicits a reply.

TYPE: Integer

OPTIONAL PARAMETERS: None

CONDITION CODES: CCE, CCL

ERROR CODE: 67

SPECIAL ATTRIBUTES: None

OPTIONAL CAPABILITY: None

TEXT DISCUSSION: 8-16

DOUBLE PROCEDURE

PROCTIME;

OPTION EXTERNAL;

FUNCTION: *Returns a process' accumulated central processor-time.*

TYPE: *Double*

OPTIONAL PARAMETERS: *None*

CONDITION CODE: *None*

ERROR CODE: *42*

SPECIAL ATTRIBUTE: *\$*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *8-18*

PROCEDURE

PTAPE (filenum1, filenum2);

VALUE *filenum1, filenum2;*

INTEGER *filenum1, filenum2;*

OPTION EXTERNAL;

FUNCTION: *Accepts input from paper tapes not containing X-OFF control characters.*

TYPE: *None*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *CCE, CCG, CCL*

ERROR CODE: *191*

SPECIAL ATTRIBUTES: *\$*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *8-87*

PROCEDURE

QUIT (num);

VALUE num;

INTEGER num;

OPTION EXTERNAL;

FUNCTION: *Aborts a process.*

TYPE: *None*

OPTIONAL PARAMETERS: *None*

CONDITION CODE: *None*

ERROR CODE: *76*

SPECIAL ATTRIBUTES: *\$*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *8-47*

PROCEDURE

QUITPROG (num);

VALUE num;

INTEGER num;

OPTION EXTERNAL;

FUNCTION: Aborts a program

TYPE: None

OPTIONAL PARAMETERS: None

CONDITION CODE: 61

SPECIAL ATTRIBUTES: \$

OPTIONAL CAPABILITY: None

TEXT DISCUSSION: 8-48

INTEGER PROCEDURE

READ (message, expectedl);

VALUE *expectedl;*

ARRAY *message;*

INTEGER *expectedl;*

OPTION EXTERNAL;

FUNCTION: *Reads an ASCII string from an input device.*

TYPE: *Integer*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *CCE, CCG, CCL*

ERROR CODE: *64*

SPECIAL ATTRIBUTES: *None*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *8-13*

INTEGER PROCEDURE

READX (message, expectedl);

VALUE expectedl;

ARRAY message;

INTEGER expectedl;

OPTION EXTERNAL;

FUNCTION: Reads an ASCII string from an input device.

TYPE: Integer

OPTIONAL PARAMETERS: None

CONDITION CODES: CCE, CCG, CCL

ERROR CODE: 64

SPECIAL ATTRIBUTES: None

OPTIONAL CAPABILITY: None

TEXT DISCUSSION: 8-13

LOGICAL PROCEDURE

RECEIVEMAIL (pin, location, waitflag);

VALUE *pin, waitflag;*

INTEGER *pin;*

LOGICAL *waitflag;*

ARRAY *location;*

OPTION EXTERNAL;

FUNCTION: *Receives mail from another process.*

TYPE: *Logical*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *CCE, CCG, CCL*

ERROR CODE: *108*

SPECIAL ATTRIBUTES: *None*

OPTIONAL CAPABILITY: *Process-Handling*

TEXT DISCUSSION: *11-20*

PROCEDURE

RESETCONTROL

OPTION EXTERNAL;

FUNCTION: *Resets terminal to accept CONTROL-Y signal.*

TYPE: *None*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *CCE, CCL*

ERROR CODE: *55*

SPECIAL ATTRIBUTES: *None*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *8-34*

PROCEDURE

RESETDUMP;

OPTION EXTERNAL;

FUNCTION: *Disables Abort Stack Dump facility*

TYPE: *None*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *CCE, CCG*

ERROR CODE: *79*

SPECIAL ATTRIBUTES: *None*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *8-65*

INTEGER PROCEDURE

SEARCH (target, length, dict, defn);

VALUE length;

BYTE ARRAY target, dict;

INTEGER length;

BYTE POINTER defn;

OPTION VARIABLE, EXTERNAL;

FUNCTION: Searches an array for specified entry or name

TYPE: Integer

OPTIONAL PARAMETERS: defn

CONDITION CODES: None

ERROR CODE: 70

SPECIAL ATTRIBUTES: None

OPTIONAL CAPABILITY: None

TEXT DISCUSSION: 8-22

LOGICAL PROCEDURE

SENDMAIL (pin, count, location, waitflag);

VALUE *pin, count, waitflag;*

INTEGER *pin, count;*

LOGICAL *waitflag;*

ARRAY *location;*

OPTION EXTERNAL;

FUNCTION: *Sends mail to another process.*

TYPE: *Logical*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *CCE, CCG, CCL*

ERROR CODE: *107*

SPECIAL ATTRIBUTES: *None*

OPTIONAL CAPABILITY: *Process Handling*

TEXT DISCUSSION: *11-18*

PROCEDURE

SETDUMP (flags);

VALUE flags;

LOGICAL flags;

OPTION EXTERNAL;

FUNCTION: Enables stack dump on abort.

TYPE: None

OPTIONAL PARAMETERS: None

CONDITION CODES: CCE, CCG

ERROR CODE: 78

SPECIAL ATTRIBUTES: None

OPTIONAL CAPABILITY: None

TEXT DISCUSSION: 8-65

PROCEDURE

SETJCW (word);

VALUE *word;*

LOGICAL *word;*

OPTION EXTERNAL;

FUNCTION: *Sets bits in job control word.*

TYPE: *None*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *None*

ERROR CODE: *72*

SPECIAL ATTRIBUTES: *None*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *8-69*

PROCEDURE

STACKDUMP (filen, idnumber, flags, selec);

BYTE ARRAY filen;

LOGICAL flags;

INTEGER idnumber;

DOUBLE ARRAY selec;

OPTION VARIABLE, EXTERNAL;

FUNCTION: *Dumps selected parts of stack to any file*

TYPE: *None*

OPTIONAL PARAMETERS: *filen, idnumber, flags, selec*

CONDITION CODES: *CCE, CCG, CCL*

ERROR CODE: *77*

SPECIAL ATTRIBUTES: *None*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *8-59*

PROCEDURE

SUSPEND (susp, rin);

VALUE *susp, rin;*

LOGICAL *susp;*

INTEGER *rin;*

OPTION VARIABLE, EXTERNAL;

FUNCTION: *Suspends a process.*

TYPE: *None*

OPTIONAL PARAMETERS: *rin*

CONDITION CODES: *CCE, CCL*

ERROR CODE: *103*

SPECIAL ATTRIBUTE: *\$*

OPTIONAL CAPABILITY: *Process Handling*

TEXT DISCUSSION: *11-13*

LOGICAL PROCEDURE

SWITCHDB (index);

VALUE index;

LOGICAL index;

OPTION PRIVILEGED, EXTERNAL;

FUNCTION: Switches DB-register pointer.

TYPE: Logical

OPTIONAL PARAMETERS: None

CONDITION CODES: CCE, CCL

ERROR CODE: 139

*SPECIAL ATTRIBUTES: **

OPTIONAL CAPABILITY: Privileged Mode

TEXT DISCUSSION: 14-5

PROCEDURE

TERMINATE;

OPTION EXTERNAL;

FUNCTION: Terminates a process.
TYPE: None
OPTIONAL PARAMETERS: None
CONDITION CODES: None
ERROR CODE: 60
SPECIAL ATTRIBUTES: \$
OPTIONAL CAPABILITY: None
TEXT DISCUSSION: 8-47, 11-14

DOUBLE PROCEDURE

TIMER;

OPTION EXTERNAL;

FUNCTION: *Returns system-timer bit-count.*

TYPE: *Double*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *None*

ERROR CODE: *40*

SPECIAL ATTRIBUTES: *\$*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *8-17*

PROCEDURE

UNLOADPROC (procid);

VALUE *procid;*

LOGICAL *procid;*

OPTION EXTERNAL;

FUNCTION: *Dynamically unloads a library procedure.*

TYPE: *None*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *CCE, CCL*

ERROR CODE: *81*

SPECIAL ATTRIBUTES: *None*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *7-29*

PROCEDURE

UNLOCKGLORIN (rinnum);

VALUE rinnum;

LOGICAL rinnum;

OPTION EXTERNAL;

FUNCTION: Unlocks a global RIN.

TYPE: None

OPTIONAL PARAMETERS: None

CONDITION CODES: CCE, CCG, CCL

ERROR CODE: 35

SPECIAL ATTRIBUTES: None

OPTIONAL CAPABILITY: None

TEXT DISCUSSION: 9-4

PROCEDURE

UNLOCKLOCIN (rinnum);

VALUE rinnum;

LOGICAL rinnum;

OPTION EXTERNAL;

FUNCTION: Unlocks a local RIN.

TYPE: None

OPTIONAL PARAMETERS: None

CONDITION CODE; CCE, CCG, CCL

ERROR CODE: 33

SPECIAL ATTRIBUTES: None

OPTIONAL CAPABILITY: None

TEXT DISCUSSION: 9-8

PROCEDURE

| |
|---------------------------------------------------------------------------|
| <i>WHO</i> (mode, capability, lattr, usern, groupn, acctn, homen, termn); |
|---------------------------------------------------------------------------|

LOGICAL mode, termn;

DOUBLE capability, lattr;

BYTE ARRAY usern, groupn, acctn, homen;

OPTION VARIABLE, EXTERNAL;

FUNCTION: Returns user attributes.

TYPE: None

OPTIONAL PARAMETERS: mode, capability, lattr, usern, groupn, acctn, homen, termn

CONDITION CODES: None

ERROR CODE: 69

SPECIAL ATTRIBUTES: None

OPTIONAL CAPABILITY: None

TEXT DISCUSSION: 8-19

PROCEDURE

XARITRAP (mask,plabel,oldmask,oldplabel);

VALUE *mask, plabel;*

INTEGER *mask, plabel, oldmask, oldplabel;*

OPTION EXTERNAL;

FUNCTION: *Arms the software arithmetic trap.*

TYPE: *None.*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *CCE, CCG, CCL*

ERROR CODE: *50*

SPECIAL ATTRIBUTES: *None*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *8-31*

PROCEDURE

XCONTRAP (plabel, oldplabel);

VALUE *plabel;*

INTEGER *plabel, oldplabel;*

OPTION EXTERNAL;

FUNCTION: *Arms or disarms the control-Y trap.*

TYPE: *None*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *CCE, CCG, CCL*

ERROR CODE: *54*

SPECIAL ATTRIBUTES: *None*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *8-36*

PROCEDURE

XLIBTRAP (plabel, oldplabel);

VALUE *plabel;*

INTEGER *plabel, oldplabel;*

OPTION EXTERNAL;

FUNCTION: *Arms or disarms the software library trap.*

TYPE: *None*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *CCE, CCG, CCL*

ERROR CODE: *52*

SPECIAL ATTRIBUTES: *None*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *8-33*

PROCEDURE

XSYSTRAP (plabel, oldlabel);

VALUE *plabel;*

INTEGER *plabel, oldplabel;*

OPTION EXTERNAL;

FUNCTION: *Arms or disarms the system trap.*

TYPE: *None*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *CCE, CCG, CCL*

ERROR CODE: *53*

SPECIAL ATTRIBUTES: *None*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *8-34*

INTEGER PROCEDURE

ZSIZE (size);

VALUE *size;*

INTEGER *size;*

OPTION EXTERNAL;

FUNCTION: *Changes size of Z-DB area.*

TYPE: *Integer*

OPTIONAL PARAMETERS: *None*

CONDITION CODES: *CCE, CCG, CCL*

ERROR CODE: *136*

SPECIAL ATTRIBUTES: *None*

OPTIONAL CAPABILITY: *None*

TEXT DISCUSSION: *8-45*

APPENDIX D

Intrinsic Error Numbers

The following listing shows intrinsic error numbers versus the intrinsics to which they pertain:

| Error Number | Intrinsic | Error Number | Intrinsic |
|---------------------|------------------|---------------------|------------------|
| 1 | FOPEN | 42 | PROCTIME |
| 2 | FREAD | 50 | XARITRAP |
| 3 | FWRITE | 51 | ARITRAP |
| 4 | FUPDATE | 52 | XLIBTRAP |
| 5 | FSPACE | 53 | XSYSTRAP |
| 6 | FPOINT | 54 | XCONTRAP |
| 7 | FREADDIR | 55 | RESETCONTROL |
| 8 | FWRITEDIR | 56 | CAUSEBREAK |
| 9 | FCLOSE | 60 | TERMINATE |
| 10 | FCHECK | 61 | QUITPROG |
| 11 | FGETINFO | 62 | BINARY |
| 12 | FREADSEEK | 63 | ASCII |
| 13 | FCONTROL | 64 | READ, READX |
| 14 | FSETMODE | 65 | PRINT |
| 15 | FLOCK | 66 | PRINTOP |
| 16 | FUNLOCK | 67 | PRINTOPREPLY |
| 17 | FRENAME | 68 | COMMAND |
| 18 | FRELATE | 69 | WHO |
| 19 | FREADLABEL | 70 | SEARCH |
| 20 | FWRITELABEL | 71 | MYCOMMAND |
| 30 | GETLOCRIN | 72 | SETJCW |
| 31 | FRELOCRIN | 73 | GETJCW |
| 32 | LOCKLOCRIN | 74 | DBINARY |
| 33 | UNLOCKLOCRIN | 75 | DASCII |
| 34 | LOCKGLORIN | 76 | QUIT |
| 35 | UNLOCKGLORIN | 77 | STACKDUMP |
| 40 | TIMER | 78 | SETDUMP |
| 41 | CHRONOS | 79 | RESETDUMP |

| Error Number | Intrinsic | Error Number | Intrinsic |
|--------------|-------------|--------------|-------------|
| 80 | LOADPROC | 110 | GETPROCINFO |
| 81 | UNLOADPROC | 112 | GETPROCID |
| 82 | INITUSLF | 120 | GETPRIORITY |
| 83 | ADJUSTUSLF | 130 | GETDSEG |
| 84 | EXPANDUSLF | 131 | FREEDSEG |
| 99 | DEBUG | 132 | DMOVIN |
| 100 | CREATE | 133 | DMOVEOUT |
| 102 | KILL | 134 | ALTDSEG |
| 103 | SUSPEND | 135 | DLSIZE |
| 104 | ACTIVATE | 136 | ZSIZE |
| 105 | GETORIGIN | 139 | SWITCHDB |
| 106 | MAIL | 191 | PTAPE |
| 107 | SENDMAIL | 200 | GETPRIVMODE |
| 108 | RECEIVEMAIL | 201 | GETUSERMODE |
| 109 | FATHER | 202 | CTRANSLATE |
| | | None | CHECKDEV |
| | | None | CLOCK |
| | | None | CALENDAR |

APPENDIX E

Disc File Labels

Whenever a disc file is created, MPE/3000 automatically supplies a file label in the first sector of the first extent occupied by that file. Such labels always appear in the format described below. (User-supplied labels, if present, are located in the sectors immediately following the MPE/3000 file label.) The contents of a label may be listed by using the *:LISTF -1* command described in Section V.

| 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 | 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. |
| | } Dates in same format as return value for CHRONOS intrinsic (Section VIII.) |
| 26 | File code. |
| 27 | File control block vector. |
| 28 | Store Bit. (If on, :STORE or :RESTORE, in progress.) |
| | (Bit 1:1) Restore Bit. (If on, :RESTORE in progress.) |
| | (Bit 2:1) Load Bit. (If on, program file is loaded.) |
| | (Bit 3:1) Exclusive Bit. (If on, file is opened with exclusive access.) |

Words

Contents

| | | |
|-------|-------------|------------------------------------------|
| | (Bits 4:4) | Device sub-type. |
| | (Bits 8:6) | Device type. |
| | (Bit 14:1) | File is open for write. |
| | (Bit 15:1) | File is open for read. |
| 29 | (Bits 0:8) | Number of user labels written. |
| | (Bits 8:8) | Number of user labels. |
| 30-31 | | Maximum number of logical records. |
| 32-34 | | Not used. |
| 35 | | Cold-load identity. |
| 36 | | Foptions specifications. |
| 37 | | Logical record size (in negative bytes). |
| 38 | | Block size (in words). |
| 39 | (Bits 0:8) | Sector offset to data. |
| | (Bits 8:4) | Not used. |
| | (Bits 12:4) | Number of extents. |
| 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. |

APPENDIX F

:STORE Tape Format

The format of a tape created by the :STORE command is:

| Item No. | Item | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------|-------------------|-----|------------|--------------------------------|-------------|-------|---------------------|--|----|--------------|--|----|-----------------------------------------------------------|---|--|--|-------------------|----|---------------------------------------|---|--|--|-------------------|----|--------------------------------------------------|--|--|-------|---------------------|
| 1 | End-of-File (EOF) Mark | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 2 | EOF Mark | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 3 | Header label (40 words), used as follows: | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | <table border="0" style="width: 100%;"> <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/3000.</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" style="font-size: 3em; vertical-align: middle;">}</td> </tr> <tr> <td></td> <td></td> <td>Date of creation.</td> </tr> <tr> <td>25</td> <td>Bits (0:8) = hours (8:8) = minutes</td> <td rowspan="2" style="font-size: 3em; vertical-align: middle;">}</td> </tr> <tr> <td></td> <td></td> <td>Time of creation.</td> </tr> <tr> <td>26</td> <td>Bits (0:8) = seconds (8:8) = tenth-of-seconds</td> <td></td> </tr> <tr> <td></td> <td>27-39</td> <td>Unused by MPE/3000.</td> </tr> </tbody> </table> | Words | Contents | | 0-13 | "STORE/RESTORE LABEL-HP/3000." | | 14-22 | Unused by MPE/3000. | | 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 | } | | | Time of creation. | 26 | Bits (0:8) = seconds (8:8) = tenth-of-seconds | | | 27-39 | Unused by MPE/3000. |
| Words | Contents | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0-13 | "STORE/RESTORE LABEL-HP/3000." | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 14-22 | Unused by MPE/3000. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 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 | } | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | Time of creation. | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 26 | Bits (0:8) = seconds (8:8) = tenth-of-seconds | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | 27-39 | Unused by MPE/3000. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 4 | EOF Mark | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 5 | Tape directory — Consists of 12-word entries, blocked 85 per 1020-word record. (The last record 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="width: 100%;"> <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 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

| Item No. | Item | | | | | | |
|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------|-----|----|------------------------------------------------------------|----|------------------------------------------------------------|
| 7 | First file. The data is blocked with 1024 words per physical tape record. (The last record 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 border="0"> <thead> <tr> <th style="text-align: left;">Word</th> <th style="text-align: left;">Use</th> </tr> </thead> <tbody> <tr> <td style="padding-left: 40px;">21</td> <td>=1 means that preceding file ended with preceding EOF mark</td> </tr> <tr> <td style="padding-left: 40px;">22</td> <td>=1 means that entire tape set ends with preceding EOF mark</td> </tr> </tbody> </table> | 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 |
| 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.

APPENDIX G

End-of-File Indication

The end-of-file indication will be 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 (See Note.) |
| E | Hardware-sensed end-of-file |

NOTE: If the word count is less than 3 or the byte count is less than 6, then Type B reads are converted to Type A reads.

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 will be rejected with an end-of-file indication.

When reading from an unlabeled tape file, the request encountering a tape mark will respond with an end-of-file indication but succeeding requests will be allowed to continue to read data past the tape mark. Under these conditions, it is the responsibility of the caller to protect against the occurrence of data beyond an end-of-file and to prevent reading off the end of the reel.



Index

A

Abnormal termination, 8-47, 8-48
Abort, 3-23, 10-17–10-21
:ABORT command, 3-23, B-1
Account
 definition of, 2-23
 name, 2-23, 3-11
 password, 2-23, 3-11, 3-19–3-21
Account Librarian Attribute, 2-27
Account Manager Capability, 2-24, 2-25, 2-27
Accounting information, printing of, 8-2
Accounting System
 purpose of, 1-5, 1-7
 use in limiting resources, 2-24
ACTIVATE intrinsic, 11-12, C-2
Active bit, 11-8
Active substate, 11-3–11-5
Actual file designators, 4-1, 4-2, 5-4
-ADDRL command, 7-19, B-7
-ADDSL command, 7-24, B-7
ADJUSTUSLF intrinsic, 8-72, 8-73, C-3
Alive state, 11-3–11-5
Allocated (loaded) program listing, 4-18, 4-21
Allocation
 and execution, 2-13, 4-6
 definition of, 2-13, 4-6
ALTDSEG intrinsic, 12-7, C-4
:ALTSEC command, 5-51, B-1
American Standard Code for Information
 Interchange, 1-2, A-1–A-5
Append-access, 5-15, 6-25
ARITRAP intrinsic, 8-31, C-5
Arrays, searching of, 8-22–8-24, 8-26
AS subqueue, 2-11, 3-13, 14-6–14-9
ASCII character set, 1-2, A-1–A-5
ASCII intrinsic, 8-11, C-6
ASCII/binary number conversion, 8-10–8-13
Assigned Storage List, 2-21
Attribute
 Account Librarian, 2-27
 Account Manager, 2-24, 2-25, 2-27
 Batch-Access, 2-29, 4-14
 Capability-Class, 2-28, 4-11, 4-18, 7-14, 8-20
 Data-segment Management, 2-28, 12-1–12-7

 File access, 2-28
 Group Librarian, 2-27
 Interactive-Access, 2-29, 4-14
 Multiple-RIN, 2-28, 13-1, 13-2
 Privileged-mode, 2-28, 4-17, 4-21, 11-8, 14-1–14-6
 Process-handling, 2-28, 11-1–11-29
 System Manager, 2-24, 2-25, 2-27
 System Supervisor, 2-27, 5-36, 5-40, 8-2
 User, 2-27, 8-19
-AUXUSL command, 7-9, B-7
Available Space List, 2-21

B

Back-referencing files, 4-3, 4-4, 5-5, 5-10
Back-up of files, 5-35, 5-43
:BASIC command, 4-5, B-1
:BASICGO command, 4-11, B-1
:BASICOMP command, 4-7, B-1
:BASICPREP command, 4-9, B-1
BASIC/3000, 1-2, 4-5
Batch-Access Attribute (Capability), 2-29, 4-14
Batch processing, 1-1, 1-2, 1-3, 1-8–1-10, 3-10
BINARY intrinsic, 8-10, C-7
Binary/ASCII Number Conversion, 8-11, 8-12
Blockfactor, 5-14, 6-18, 6-27, 6-29
Bounds check, 2-14, 3-9, 3-10
BREAK key, 3-21, 8-46, 8-76, 8-83
Breakable commands, B-6
Breakpoints
 clearing, 8-51
 resuming execution after, 8-51
 setting, 8-48–8-50
BS subqueue, 2-11, 3-13, 14-7–14-9
Buffers, input/output, 5-16, 6-8
:BUILD command, 5-30, B-2
-BUILDRL command, 7-18, B-7
-BUILDSL command, 7-23, B-7
-BUILDUSL command, 7-3, B-7
:BYE command, 3-23, 3-24, B-2

C

- CALENDAR intrinsic, 8-18, C-10
- Callable intrinsics, 2-8
- Capability-Class Attributes, 2-28, 4-11, 4-18, 7-14, 8-20
- Capability sets, 1-3, 1-6, 2-25-2-29
- Carriage-control characters, 5-15, 6-22, 6-33, 6-37-6-39, 6-56
- CAUSEBREAK intrinsic, 8-46, 8-83
- CEASE command, 7-6, B-5
- Central processor time
 - accounting of, 1-5
 - limit on, 3-13, 3-22, 3-25, 3-26
 - reporting of, 3-23, 8-2
- Changing input echo facility, 8-81
- Changing size of data segment, 12-7
- Changing terminal characteristics, 8-74-8-91
- Changing terminal speed, 8-78-8-81
- CHECKDEV intrinsic, 8-70, C-9
- CHRONOS intrinsic, 8-18, C-10.1
- Circular Subqueues, 2-11, 14-6-14-9
- CLOCK intrinsic, 8-18, C-10.2
- Closing files, 6-1, 6-30-6-32
- COBOL/3000, 1-2, 4-7, 4-8, 4-9, 4-11, B-1
 - :COBOL command, 4-7, 4-8, B-1
 - :COBOLGO command, 4-11, B-2
 - :COBOLPREP command, 4-9, B-2
- Code
 - re-entrancy of, 1-3
 - segments, 1-1, 1-3, 2-2
 - sharability of, 1-3, 2-2, 2-3
- Code segments
 - in a process, 2-2
 - interaction with PCB and stack, 2-14-2-20
 - re-entrancy of, 1-3
 - sharability of, 2-2, 2-3
 - table (CST), 2-13, 2-20, 2-21, 11-3
- Command
 - continuation character for, 3-2, 3-4
 - definition of, 2-6, 3-1
 - delimiters, 3-2
 - dictionary, 8-23
 - errors, 3-6
 - format of, 3-1, 3-2, 3-4, 3-5
 - executors, 2-8
 - parameter, keyword, 3-3
 - parameter formatting, programmatic, 8-25
 - parameter list for, 3-2
 - parameter positional, 3-3
 - programmatic invoking of, 3-1, 8-28-8-30
 - purposes of, 2-6
- Command Interpreter
 - programmatic invoking of, 3-1
 - purpose of, 1-6
 - use in command translation, 2-6, 2-21, 2-23
- COMMAND intrinsic, 8-29, C-11
- :COMMENT command, 8-9, B-2
- Compilation
 - and preparation, 4-9-4-11
 - and preparation/execution, 4-11
 - definition and function, 2-11, 4-6
 - invoking COBOL/3000, 4-7, 4-9, 4-10

- invoking FORTRAN/3000, 4-7, 4-9, 4-10
- invoking SPL/3000, 4-7, 4-9, 4-10
- Compilers
 - COBOL/3000, 1-2, 4-7-4-9, 4-11, B-1
 - FORTRAN/3000, 1-2, 4-7-4-11
 - intrinsics for use in writing, 8-71-8-74
 - SPL/3000, 1-2, 4-7-4-9, 4-11, B-3
- Compute-bound processes, 2-11
- Condition codes, 3-9
- Continuation character, command, 3-2, 3-4
- :CONTINUE command, 3-9, 3-26, 8-67, B-2
- COPY command, 7-9, B-5
- Connect time, terminal
 - accounting of, 1-5
 - limit on, 3-22
 - reporting of, 3-16, 3-22, 8-2
- CREATE intrinsic, 11-7, 14-9, C-12
- Creating a new file, 5-30, 6-3
- Creating a process, 11-7, 14-9
- Creating an extra data segment, 12-1, 12-2
- Creating files, 5-30, 6-1, 6-3-6-29
- CS subqueue, 2-11, 3-13, 14-7-14-9
- CTranslate intrinsic, 8-13, 8-13A, C-12.1

D

- DAScii intrinsic, 8-13, C-13
- Data
 - access of, 3-25, 3-26
 - input with :DATA command, 3-25, 3-26
 - termination of, 3-16
 - :DATA command, 3-25, 3-26, 6-5, B-2
 - Data-base register, 2-16, 2-17
 - Data-limit register, 2-16, 2-17, 2-18
 - Data segments
 - extra acquisition of, 12-1-12-3
 - extra, and privileged-mode users, 14-6
 - extra, changing size of, 12-7
 - extra, creation of, 12-1, 12-2
 - extra, definition of, 12-1
 - extra, deletion of, 12-3, 12-4
 - extra, moving data from, 12-4, 12-5, 14-4
 - extra, moving data into, 12-5, 12-6
 - extra, release of, 12-3, 12-4
 - management of, 12-1
 - privacy of, 1-3
 - table (DST), 2-13, 2-20, 2-21, 11-5
 - Data segment management capability, 2-28, 12-1-12-7
 - Date, printing of, 8-2, 8-18
 - DB-Q initial area of stack, 2-16, 2-17
 - DBINARY intrinsic, 8-11, C-14
 - DB register, 2-16, 2-17
 - Deadlock, 11-21, 13-1, 13-2
 - DEBUG commands, 8-50, 8-51
 - DEBUG intrinsic, 8-48-8-67
 - DEBUG privileged mode, 14-10
 - Deleting files, 5-32
 - Delimiters, command, 3-2
 - device characteristics, 8-91

- devicefile, 2-14, 6-5-6-14
- device recognition, 6-6
- Devices, diagnostic, verifying assignment of, 8-70
- Diagnostic devices, 8-70
- Diagnostician Attribute, 2-26
- Dictionary, command, 8-22
- Direct-access files, 1-3, 6-35, 6-41
- Disc extents, 5-2, 5-18, 6-3, 6-20
- Dispatcher
 - in process creation, 11-5
 - purpose of, 1-6
 - use in searching master queue, 2-9
- DL-DB area of stack, 2-16, 2-17, 4-13, 4-18, 4-22, 7-14, 8-44, 8-45, 11-10
- DL register, 2-16, 2-17, 2-18
- DLSIZE intrinsic, 8-44, 8-45, C-16
- DMOVIN intrinsic, 12-4, 12-5, C-17
- DMOVOUT intrinsic, 12-5, 12-6, C-18
- Domains, File, 2-24, 5-3, 5-4, 5-16, 6-10
- DS subqueue, 2-11, 3-13, 14-7-14-9
- Duplex mode, 8-81-8-83
- Duplicative pairs, 6-62, 6-63, 8-66
- Dynamic loading of procedures, 7-27-7-30

E

- Echo facility, 8-81-8-83
- EDIT/3000, 4-23
- :EDITOR command, 4-23, B-2
- End-of-file indicator, 6-56-6-58, G-1
- :EOD command, 3-16, 3-17, 3-25, B-2
- :EOJ command, 3-16, B2
- Errors
 - abort, 3-9
 - and file information display, 10-31, 10-34
 - checking of, 6-52-6-55
 - command interpreter, 3-6
 - condition code, 3-9, 6-2
 - numbers, D-1, D-2
 - recovery from, 3-6, 6-58
 - run-time, 10-1, 10-16, 10-29
- ES subqueue, 2-11, 3-13, 14-7-14-9
- ESC: function, 3-13, 8-76
- ESC; function, 3-13, 8-76
- Execution
 - definition of, 2-13, 4-6
 - of programs, 4-21
- EXIT command, 7-2, B-5
- EXPANDUSLF intrinsic, 8-72, 8-73, C-19
- Extents file, 5-2, 5-18, 6-3, 6-20
- External procedure libraries, 7-15-7-26
- Extra data segments
 - acquisition of, 12-1-12-3
 - and privileged-mode users, 14-6
 - changing size of, 12-7
 - creation of, 12-1, 12-2
 - definition of, 12-1
 - deletion of, 12-3, 12-4
 - moving data from, 12-4, 12-5, 14-4

- moving data into, 12-5, 12-6
- release of, 12-3, 12-4
- management of, 12-1
- privacy of, 1-3

F

- FATHER intrinsic, 11-25, C-20
- Father process, 2-2, 11-25
- FCHECK intrinsic, 6-52-6-55, 8-85, C-21
- FCLOSE intrinsic, 6-30-6-32, C-22
- FCONTROL intrinsic, 6-56, 8-80-8-91, C-23
- FGETINFO intrinsic, 6-48-6-51, C-24
- File
 - access information, return of, 6-48-6-51
 - access modes, 5-45-5-53
 - access of, 5-4, 5-15, 5-16, 5-20, 6-25
 - actual designators for, 4-1, 4-2, 5-4
 - aoptions, 6-17, 6-25-6-29
 - back-reference, 4-3, 4-4, 5-5, 5-10
 - back-up of, 5-35-5-43
 - blockfactor, 5-14, 6-18, 6-27, 6-29
 - buffers for, 5-16, 6-19
 - carriage-control characters, 5-15, 6-22, 6-33, 6-37-6-39, 6-56
 - changing security provisions, 5-51
 - characteristics, 5-1, 5-10-5-28
 - closing, 6-1, 6-30-6-32
 - command summary, 5-55
 - control blocks for, 6-3
 - control operations, direction of, 6-56-6-58
 - creation of, 5-30
 - default assignment of, 4-1, 4-4
 - definition of, 2-9
 - deleting, 5-32
 - devices for, 1-3, 2-9, 5-2, 5-3, 5-18, 6-3, 6-4, 6-18
 - direct-access, 1-3, 6-39, 6-41
 - disc, 1-3
 - disc extents for, 5-2, 5-18, 6-3, 6-20
 - disposition of, 6-2, 6-31
 - domain, job/session, 5-4, 5-16, 6-21
 - domain, system, 5-3, 5-16, 6-21
 - dumping off-line, 5-35-5-40
 - duplicative pairs, 6-61, 6-62, 8-51
 - end-of-file indicator, 6-56-6-58, G-1
 - error-checking, 6-52-6-55
 - error recovery for tape, automatic, 6-58
 - exclusive use of, 1-3, 5-16, 6-26
 - filecode, 5-18, 6-20
 - filereference format, 4-3, 4-4, 5-6, 5-7-5-9
 - foptions, 6-17, 6-20-6-25
 - formal designators for, 4-2, 4-4, 5-4, 5-5, 5-26-5-28, 6-17
 - hardware status word, 6-56-6-58
 - implicit :FILE commands for, 5-25
 - information display, 10-31-10-34
 - input/output sets, 4-4, 5-10
 - input/output verification, 6-56-6-58
 - interactive pairs, 6-61, 6-62, 8-15

intrinsic summary, 6-63
 job/session input device, 4-4
 job/session listing device, 4-4
 job/session temporary, 4-3, 4-4, 5-6
 labels, 6-3, 6-18, 6-43, 6-44, E-1, E-2
 locking of, 5-54, 6-26, 6-60, 9-9-9-11
 lockword, 5-3, 5-7-5-9
 management of, 1-6
 multi-access of, 5-16, 5-18-5-20, 6-26
 names of, 1-3, 2-9, 5-7
 naming as positional parameters in command, 4-1
 new, 4-1, 4-3
 \$NEWPASS, 4-3, 4-4, 5-6, 5-10
 \$NULL, 4-2, 4-4, 5-5, 5-10
 number, 6-3
 old, 4-1, 4-3, 5-6
 \$OLDPASS, 4-3, 4-4, 5-6, 5-10, 5-31
 opening, 6-1, 6-3-6-29
 passing of, 4-3, 4-4, 5-6, 5-10
 permanent, 4-3, 4-4, 5-16, 5-31
 permanent, accounting of space for, 1-5, 5-9, 8-2
 purging, 5-32
 reading direct access, 6-34, 6-36
 reading labels of, 6-43
 reading sequential, 6-32
 referencing, 4-1-4-4
 releasing security provisions, 5-53
 renaming of, 5-44, 6-60
 re-setting formal designators for, 5-11, 5-29, 5-31
 resetting logical record pointer, 6-47
 re-specifying names of, 5-21
 restoring security provisions, 5-54
 retrieval of dumped, 5-40-5-44
 rewinding, 6-56-6-58
 saving, 5-31
 sectors required by, 5-2
 security provisions, 2-25, 5-45-5-54, 6-3, 6-32
 sequential, 1-3, 6-32, 6-33, 6-37-6-41
 sets, listing of, 5-32
 sharability of, 1-3
 size of, 5-18, 6-19
 spacing forward or backward on, 6-46
 status information, return of, 6-48
 \$STDIN, 4-2, 4-4, 5-5, 5-10
 \$STDINX, 4-2, 4-4, 5-5, 5-10
 \$STDLIST, 4-2, 4-4, 5-5, 5-10
 subsystem formal designators, 5-26-5-28
 system-defined, 4-1, 4-2, 4-4, 5-4, 5-5
 tape mark, 6-56-6-58
 temporary, 5-16
 time out interval for terminal, 6-56-6-58
 terminal control, 6-58
 updating, 6-45
 user pre-defined, 4-1, 4-2, 5-5
 writing labels for, 6-44
 writing to direct access, 6-41
 writing to sequential access, 6-37-6-41
 File-Access Attributes, 2-28
 :FILE command, 5-11-5-28, 6-22, B-2
 File Management System, 1-6, 2-9

Fixed-length records, 1-3, 5-15, 6-15
 FLOCK intrinsic, 6-60, 9-9, 9-10, 13-1, C-25
 FOPEN intrinsic, 6-1, 6-17-6-30, C-26
 Formal file designators, 4-2, 4-4, 5-4, 5-5, 5-26-5-28, 6-17
 forms alignment, 6-9
 :FORTGO command, 4-11, B-2
 :FORTPREP command, 4-9, 4-10, B-2
 :FORTRAN command, 4-7, 4-8, B-2
 FORTRAN/3000, 1-2, 4-7-4-11
 FPOINT intrinsic, 6-47, C-27
 FREAD intrinsic, 6-32, 6-33, C-28
 FREADDIR intrinsic, 6-34, C-29
 FREADLABEL intrinsic, 6-43, C-30
 FREADSEEK intrinsic, 6-36, C-31
 FREEDSEG intrinsic, 12-3, 12-4, C-32
 FREELOCIN intrinsic, 9-9, C-33
 FRELATE intrinsic, 6-61, 6-62, C-34
 :FREERIN command, 9-6, B-2
 FRENAME intrinsic, 6-60, 6-61, C-34
 FSETMODE intrinsic, 6-58-6-60, C-36
 FSPACE intrinsic, 6-46, C-37
 FUNLOCK intrinsic, 6-60, 9-11, C-38
 FUPDATE intrinsic, 6-45, C-39
 FWRITE intrinsic, 6-37-6-41, C-40
 FWRITEDIR intrinsic, 6-41, C-41
 FWRITELABEL intrinsic, 6-44, C-42

G

GETDSEG intrinsic, 12-1-12-3, 14-5, 14-6, C-43
 GETJCW intrinsic, 8-69, C-44
 GETLOCIN intrinsic, 9-6, C-45
 GETORIGIN intrinsic, 11-25, C-46
 GETPRIORITY intrinsic, 11-23, 11-24, 11-26, 14-9, C-47
 GETPRIVMODE intrinsic, 14-3, C-48
 GETPROCID intrinsic, 11-26, C-49
 GETPROCINFO intrinsic, 11-28, C-50
 :GETRIN command, 9-21, B-2
 GETUSERMODE intrinsic, 14-4, C-51
 Global area of stack, 2-16, 2-17
 Global RIN, 9-2-9-6
 Group
 definition of, 2-24
 home, 2-25, 3-11
 name, 2-24, 3-11
 password, 2-24, 3-11, 3-12, 3-19-3-21
 public, 2-24
 Group Librarian Attribute, 2-27

H

Half-duplex mode, 3-20, 8-81, 8-82
 Hardware status word, 6-56, 6-58
 :HELLO command, 3-19-3-22, B-3
 -HIDE command, 7-27, B-5
 Home group, 2-25, 3-11

HP 3000 Computer, 1-1, 1-2, 1-5

I

Implicit :FILE commands, 5-25
INITUSLF intrinsic, 8-71, 8-73, C-52
Input/output-bound processes, 2-11
Input/output sets, 4-4, 5-10
Input/Output System
 controller, 2-8
 purpose of, 1-6, 2-8
 relationship to File Management System, 2-8
 relationship to Memory Management System, 2-8
Interactive Attribute (Capability), 2-29, 4-14
Interactive pairs, 6-61, 6-62, 8-19
Interactive processing, 1-2, 1-3, 1-8-1-10, 3-19-3-24
 4-14
Internal flag, 7-27
Interpreter, BASIC/3000, 1-2, 4-5
Interpreter, Command, 1-6, 2-6, 2-21, 2-23, 3-1, 8-28-
 8-30
Intrinsic
 callable, 2-8
 condition code returned by, 3-9
 definition of, 2-6
 error numbers, D-1, D-2
 format, 3-8
 uncallable, 2-8, 14-1
Intrinsic calls
 definition of, 2-6, 3-1
 errors, 3-9
 format, 2-8, 3-9
 issuing from higher-level languages, 3-1, 3-6, 6-1
 issuing from SPL/3000, 3-1, 3-6, 3-7, 6-1
INTRINSIC declaration, 3-7

J

Job
 control word, 8-67, 8-69
 definition of, 1-1
 identification, 3-25
 initiation of, 3-10
 processing of, 1-2, 2-20
 status report, 8-2-8-4
 structure of, 3-16-3-17
 termination of, 3-16-3-26
 versus session, 1-2
:JOB command, 3-10-3-14, B-3
Job Main Process, 2-5, 2-21, 2-22
Job/session input device
 definition of, 3-10
 filename for, 4-4
 reading ASCII characters from, 8-13, 8-14
Job/session listing device
 definition of, 3-10
 filename for, 4-4
 writing output to, 8-15
Job/session temporary files, 4-3, 4-4, 5-6

K

keyword parameters, 3-3
KILL intrinsic, 11-16, C-53

L

Labels, File, 6-3, 6-18, 6-43, 6-44, E-1, E-2
Librarian
 account, 2-27
 group, 2-27
Library procedures, 7-15-7-26
Linear subqueues, 2-11, 14-6-14-9
:LISTF command, 5-32, B-3
-LISTRL command, 7-20, B-5
-LISTSL command, 7-24, B-5
-LISTUSL command, 7-10, B-5
LMAP, 4-18-4-21
Loader, 2-23
LOADPROC intrinsic, 7-28, C-54
Local area, 2-17, 4-13, 4-18, 4-23, 7-14, 11-9
Local Attributes, 2-29
Local RIN's, 9-6-9-9
LOCKGLORIN intrinsic, 9-3, 13-1, C-55
LOCKLOCRIN intrinsic, 9-7, C-56
Lockwords, 5-3, 5-7, 5-9
Logging Facility, 1-5, 1-7
Logical records, 2-9, 5-1, 5-2, 6-5
Logical record pointer, 6-47

M

Mail, definition of, 11-16
MAIL intrinsic, 11-17, C-57
Main memory
 linkages, 2-20, 2-21
 maximum size of, 1-4
 use of, 2-2, 2-21
Manager
 Account, Attributes, 2-24, 2-25, 2-27
 System, Attribute, 2-24, 2-25, 2-27
Map
 of allocated (loaded) program, 4-18-4-21
 of prepared program, 4-17
Master Scheduling Queue, 2-9
Memory, main, 1-4, 2-2, 2-20, 2-21
Memory, virtual, 1-4, 2-2
Memory Management System
 control of swapping by, 2-20
 purpose of, 1-6
Messages
 command interpreter error, 10-1-10-14
 command interpreter warning, 10-1, 10-14, 10-15
 console operator, 10-1, 10-29, 10-30
 file information display, 10-31-10-33
 operator, 10-1, 10-29, 10-30
 run-time, 10-1, 10-16-10-29
 system, 10-1, 10-31

- system failure, 10-2
- user, format of, 8-8, 10-29
- user, transmission of, 8-8, 8-9, 10-1
- Microprogramming, 1-4
- MPE/3000
 - back-up of, 1-6
 - definition of, 1-1
 - features of, 1-1-1-7, 2-1
 - functions of, 1-1
 - initialization of, 1-7
 - minimum hardware required by, 1-7
 - modification of, 1-6
 - operation of, 1-2
 - optional hardware for, 1-7
 - residence of, 2-2
 - software components of, 1-6
 - subsystem formal designators, 5-26-5-28
 - subsystems, invoking of, 4-23-4-27
- Multiple RIN Capability, 2-28, 13-1, 13-2
- Multiprogramming, 1-1
- Multiprogramming Executive Operating System
 - back-up of, 1-6
 - definition of, 1-1
 - features of, 1-1-1-7, 2-1
 - functions of, 1-1
 - initialization of, 1-7
 - minimum hardware requirements of, 1-7
 - modification of, 1-6
 - operation of, 1-2
 - optional hardware for, 1-7
 - residence of, 2-2
 - software components of, 1-6
 - subsystem formal designators, 5-26-5-28
 - subsystems, invoking of, 4-23-4-27
- Multiple RIN's, 13-1, 13-2
- MYCOMMAND intrinsic, 8-25, C-58

N

- Name
 - account, 2-24, 3-11
 - file, 1-3, 2-9, 5-7
 - group, 2-24, 3-11
 - user, 2-24, 3-11, 3-18-3-20
- New Files, 4-1, 4-3
- \$NEWPASS, 4-3, 4-4, 5-6, 5-10
- NEWSEG command, 7-8, B-5
- Non-auto recognizing devices, 3-24
- Non-privileged mode, 4-17, 4-21, 11-8, 14-1-14-6
- \$NULL, 4-2, 4-4, 5-5, 5-10
- Number conversion, 8-10-8-13

O

- Old files, 4-1, 4-3, 5-6
- \$OLDPASS, 4-3, 4-4, 5-6, 5-10, 5-31
- Opening files, 6-1, 6-3-6-29
- Operator's Console

- purpose of, 1-7, 1-8
- writing output to, 8-16
- Optional Capabilities
 - definition of, 1-3, 2-28, 2-29
 - versus standard capabilities, 2-29

P

- P-register, 2-15
- Parameters
 - formatting, programmatic, 8-25
 - keyword, 3-3
 - list, 3-2
 - positional, 3-3
- Passing Files, 4-3, 4-4, 5-6, 5-10
- Password
 - account, 2-24, 3-11, 3-19-3-21
 - group, 2-24, 3-11, 3-19-3-21
 - user, 3-11, 3-12, 3-19-3-21
- PAUSE intrinsic, 8-18A, C-58.1
- PB-register, 2-15
- PCB, 2-2, 2-9, 2-13, 2-14, 11-3
- PCBX, 2-16
- Permanent Files, 4-3, 4-4, 5-16, 5-31
- PIN, 11-5, 11-7, 11-8, 11-25
- PMAP, 4-13, 4-15, 4-17, 7-12-7-14, 7-24
- Physical records, 1-3, 5-1, 5-2, 6-15
- PL-register, 2-14
- Positional Parameters, 3-3
- :PREP command, 4-12, 4-13, B-3
- Preparation
 - and execution, 4-17
 - definition of, 2-13
 - process of, 4-6, 4-12, 7-12
- PREPARE command, 7-12, B-5
- :PREPRUN command, 4-17, B-3
- PRINT intrinsic, 8-15, C-59
- PRINTOP intrinsic, 8-16, C-60
- PRINTOPREPLY intrinsic, 8-16, C-61
- Privileged mode, 4-17, 4-22
- Privileged-mode capability, 2-28, 4-17, 4-22, 11-8, 14-1-14-6
- Privileged programs, 14-2-14-4
- Procedure
 - calls, 2-5
 - comparison with subroutines, 2-5, 2-6
 - definition of, 2-5
 - libraries, external, 7-15-7-26
 - run-time report, 8-18
- Process
 - abort of, 8-47, 11-11
 - activation of, 11-12
 - activation source, 11-25
 - break, requesting a, 8-46
 - communication between, 8-67, 11-16-11-22
 - compute-bound, 2-11
 - control block, 2-2, 2-9, 2-13, 2-15, 11-3
 - control block extension, 2-16
 - creation of, 11-6, 11-7-11-11
 - deadlocks, 11-22, 13-1, 13-2
 - definition of, 2-2

- deletion of, 11-14, 11-15, 11-16
- father, 2-2, 11-25
- input/output, 2-2
- input/output-bound, 2-11
- job main, 2-5, 2-21, 2-22
- life-cycle, 11-1–11-7
- linking, 11-5, 11-6
- mail, definition of, 11-16
- mail, receiving (collecting), 11-20, 11-21
- mail, sending, 11-18, 11-19
- mailbox, definition of, 11-16
- organization of, 2-4
- pin, 11-5, 11-7, 11-8, 11-25
- priority, 11-10, 11-22–11-24, 11-27
- root, 2-2, 14-1–14-9
- scheduling, 11-10, 11-22–11-24
- session main, 2-5
- son, 2-2, 11-26
- state, 11-27
- suspension of, 11-13, 11-14
- system, 2-2
- termination of, 8-47, 11-8, 11-13
- testing status of, 11-17, 11-18
- user, 2-2
- user controller, 2-2, 2-5, 2-21, 2-23
- Process Control Block, 2-2, 2-9, 2-13, 2-14, 11-3
- Process Control Block Extension, 2-16
- Process-handling capability, 2-28, 11-1–11-29
- Process Identification Number, 11-5, 11-7, 11-8, 11-25
- Processing
 - batch, 1-1, 1-2, 1-3, 1-8–1-10, 3-10
 - interactive, 1-2, 1-3, 1-8–1-10, 3-19–3-24, 4-14
- PROCTIME intrinsic, 8-18, C-62
- Program
 - abort of, 3-23, 8-48
 - interruption of, 3-23
 - permanently-privileged, 14-2
 - resumption of, 3-23
 - temporarily-privileged, 14-2–14-4
- Program base register, 2-14
- Program Capability Sets, 2-29
- Program counter register, 2-15
- Program file
 - definition of, 2-13
 - use in execution, 4-21
 - use in preparation, 4-10
- Program limit register, 2-15
- Program unit, 2-11, 4-1
- Programming languages, 1-2
- Prompt character, 3-2, 7-2
- PTAPE intrinsic, 8-88, C-63
- :PTAPE command, 8-87, B-3
- Public Group, 2-23
- :PURGE command, 5-32, B-4
- PURGERBM command, 7-7, B-5
- PURGERL command, 7-19, B-6
- PURGESL command, 7-24, B-6
- Purging Files, 5-30

Q

- Q-register, 2-16, 2-17, 2-18, 2-19, 2-20, 2-21
- Queues, scheduling, 2-10, 2-11, 3-13, 14-6–14-9
- QUIT intrinsic, 8-47, C-64
- QUITPROG intrinsic, 8-48, C-65

R

- RBM
 - definition of, 2-11, 7-1
 - entry-points, 7-2, 7-27
 - in compilation, 2-11, 2-13, 4-6
 - internal flags, setting of, 7-27
- READ intrinsic, 8-14, C-66
- READX intrinsic, 8-15, C-66
- Read-access, 5-15, 6-13
- Reading Files, 6-32–6-34
- RECEIVEMAIL intrinsic, 11-20–11-22, C-68
- Record
 - blocks, 2-9, 5-1, 5-2, 5-14
 - direct-access, 5-2
 - fixed-length, 1-3, 5-15, 6-5
 - formats, 6-15, 6-22
 - logical, 2-9, 5-1, 5-2, 6-15
 - physical, 1-3, 5-1, 5-2, 6-15
 - sequential access, 5-2
 - specifying size of, 5-14, 6-18
 - undefined-length, 5-15, 6-15
 - variable-length, 1-3, 5-15, 6-15
- Registers
 - contents, display of, 8-54
 - contents, modification of, 8-56
 - DB (Data Base), 2-16, 2-17
 - DL (Data Limit), 2-16, 2-17, 2-18
 - P (Program Counter), 2-15
 - PB (Program Base), 2-15
 - PL (Program Limit), 2-15
 - Q (Stack Marker), 2-16, 2-17, 2-18, 2-19, 2-20
 - S (Top of Stack), 2-16, 2-17, 2-18, 2-19, 2-20
 - SM, 2-18
 - SR, 2-18
 - Status, 3-9
 - TR, 2-17
 - Z (Stack Limit), 2-16, 2-17, 2-18, 2-19, 2-20
- :RELEASE command, 5-53, B-4
- Relocatable binary module
 - definition of, 2-11, 7-1
 - entry-points, 7-2, 7-27
 - in compilation, 2-11, 4-6
 - internal flags, setting of, 7-27
- Relocatable Libraries
 - adding a procedure for, 7-19
 - creation of, 7-18
 - definition of, 7-15
 - deleting an entry-point from, 7-19
 - deleting a procedure from, 7-19

- designating for management, 7-19
- listing procedures in, 7-20
- Relocatable Library File, 4-15, 4-18, 7-15-7-19
- :RENAME command, 5-44, B-4
- Renaming Files, 5-44, 6-60
- :REPORT command, 8-2, B-3
- :RESET command, 5-11, 5-29, B-4
- RESETCONTROL intrinsic, 8-36, C-69
- :RESETDUMP command, 8-65, B-3
- RESETDUMP intrinsic, 8-66, C-70
- Resource identification number
 - deadlocks, 13-1
 - definition of, 9-1
 - global, 9-2-9-6
 - local, 9-6-9-9
 - locking, 9-9, 9-10
 - multiple, 13-1, 13-2
 - unlocking, 9-9, 9-11
- :RESTORE command, 5-40, B-4
- :RESUME command, 8-46, B-4
- REVEAL command, 7-27, B-6
- Rewinding Files, 6-56, 6-57
- RIN
 - deadlocks, 13-1
 - definition of, 9-1
 - global, 9-2-9-6
 - interjob level, 9-2-9-6
 - interprocess level, 9-6-9-9
 - local, 9-6-9-9
 - locking with FLOCK intrinsic, 9-9, 9-10
 - multiple, 13-1, 13-2
 - unlocking with FUNLOCK intrinsic, 9-9, 9-11
- :RJE command, 4-25, B-5
- RL
 - adding a procedure to, 7-19
 - creation of, 7-18
 - definition of, 7-15
 - deleting an entry-point from, 7-19
 - deleting a procedure from, 7-19
 - designating for management, 7-19
 - listing procedures in, 7-20
- RL File, 4-15, 4-18, 7-15-7-19
- RL command, 7-19, B-6
- Root Process, 2-2, 14-1-14-9
- :RPG command, 4-7, B-4
- :RPGGO command, 4-11, B-4
- :RPGPREP command, 4-9, B-4
- :RUN command, 4-21, B-5

S

- S-register, 2-16, 2-17, 2-18, 2-19, 2-20
- :SAVE command, 5-31, B-5
- Scheduling, 1-4, 1-6, 2-9, 2-10, 3-13, 11-10, 11-22, 11-24, 14-6-14-9
- Scheduling queues, 2-10, 2-11, 3-13, 14-6-14-9
- SDM/3000, 4-25
- SEARCH intrinsic, 8-22-8-24, 8-26, C-71
- :SECURE command, 5-53, B-5

- Security provisions, File system, 2-25, 5-45-5-54, 6-3, 6-32
- Segmented libraries
 - adding a procedure to, 7-24
 - creation of, 7-23
 - definition, 7-16
 - deleting an entry-point from, 7-24
 - deleting a segment from, 7-24
 - designating for management, 7-23
 - listing procedures in, 7-24-7-26
 - purpose of, 7-16, 7-17
- Segmenter Subsystem
 - commands, 7-2
 - exit from, 7-2
 - invoking of, 4-24, 7-2
 - purpose of, 1-7
 - use in preparation, 2-13, 4-6
- :SEGMENTER command, 4-25, 7-2, B-4
- Segments
 - code, 1-3, 2-2, 2-3, 2-14-2-21, 11-3
 - data, 1-3, 2-13, 2-20, 2-21, 11-5, 12-1-12-7, 14-4, 14-6
 - extra data, 12-1-12-7, 14-4, 14-6
- SENDMAIL intrinsic, 11-18, C-72
- Sequential Files, 1-3, 6-32, 6-33, 6-37-6-41
- Session
 - definition of, 1-2
 - identification, 3-24
 - initiation of, 3-18
 - interrupting program execution within, 3-21
 - processing of, 2-23
 - status report, 8-2-8-4
 - structure of, 3-24
 - termination of, 3-23, 3-27
 - versus job, 1-2
- Session Main Process, 2-5
- :SETDUMP command, 8-64, B-4
- SETDUMP intrinsic, 8-65, C-73
- SETJCW intrinsic, 8-69, C-74
- :SETMSG command, 8-9, B-4
- :SHOWDEV command, 8-4, B-4
- :SHOWIN command, 8-5, B-4
- :SHOWJOB command, 8-2, B-4
- :SHOWOUT command, 8-6, B-4
- :SHOWTIME command, 8-2, B-4
- SL
 - adding a procedure segment to, 7-24
 - creation of, 7-23
 - definition of, 7-16
 - deleting an entry-point from, 7-24
 - deleting a segment from, 7-24
 - designating for management, 7-23
 - listing procedures, 7-24-7-26
 - purpose of, 7-16, 7-17
- SL command, 7-23, B-6
- SM-register, 2-18
- Son Process, 2-2, 11-25
- SORT/3000, 4-23
- :SPEED command, 8-78, 8-79, B-6
- :SPL command, 4-7, 4-8, B-4

:SPLGO command, 4-11, B-4
 :SPLPREP command, 4-9, B-4
 SPL/3000, 1-2, 4-7-4-9, 4-11, B-3
 spoofer, 6-10
 spooler, 6-10
 spooling, 1-4, 2-14, 3-27, 6-10-6-14
 SR-register, 2-18
 Stack
 architecture, 1-4
 contents, display of, 8-54
 contents, modification of, 8-56-8-58
 DB-Q initial area, 2-16, 2-17
 DL-DB area, 2-16, 2-17, 4-13, 4-18, 4-23, 7-14, 8-44,
 8-45, 11-9
 definition of, 1-4
 global area (DB-Q initial), 2-16, 2-17
 in a process, 2-2
 initially-defined, 2-13
 interaction with PCB and code segments, 2-14-2-16
 local area, (Z-Q), 2-17, 4-13, 4-18, 4-22, 7-14, 11-9
 marker, 2-18, 2-19
 marker contents, trace of, 8-58
 overflow, 2-16
 privacy of, 2-2, 2-3
 stack area (Z-DL), 2-16, 2-17, 4-13, 4-18, 4-21, 7-14,
 8-44-8-46, 11-10
 top of (definition), 2-13, 2-16, 2-18
 user-managed area (DL-DB), 2-16, 2-17, 4-13, 4-18,
 4-22, 7-14, 8-44, 8-45,
 11-10
 working stack, 2-16, 2-17
 Z-DL area, 2-16, 2-17, 4-13, 4-18, 4-21, 7-14, 8-44-
 8-46, 11-10
 Z-Q area, 2-17, 4-13, 4-18, 4-22, 7-14, 11-10
 Stack-limit register, 2-16, 2-17, 2-18, 2-19, 2-20
 Stack Marker register, 2-16, 2-17, 2-18, 2-19, 2-20
 STACKDUMP intrinsic, 8-59, C-75
 Stand-Alone Diagnostic Utility Program, 4-24
 Standard capabilities, 1-2, 1-3, 2-29
 :STAR command, 4-23, B-4
 STAR/3000, 4-23
 Status register, 3-9
 \$STDIN, 4-2, 4-4, 5-5, 5-10
 \$STDINX, 4-2, 4-4, 5-5, 5-10
 \$STDLIST, 4-2, 4-4, 5-5, 5-10
 :STORE command, 5-35-5-40, B-6, F-1, F-2
 :STORE tape format, F-1, F-2
 :STREAM command, 3-28, 6-10, B-6
 Subroutines, 2-5, 2-6
 SUSPEND intrinsic, 11-13, C-76
 SWITCHDB intrinsic, 14-5, C-77
 System clock, 8-2, 8-17
 System Configurator, 1-6
 System file domain, 5-3, 6-10
 System Initiator, 1-6
 System Library, 1-6, 2-6, 7-16
 System Manager Capability
 definition and functions of, 1-2, 1-6, 2-27

in creation and modification of accounts, 2-24, 2-25
 in reporting accounting information, 8-2
 in retrieving file sets, 5-41
 in storing file sets, 5-36-5-40
 System Supervisor Capability
 definition and function of, 1-6, 2-26
 for system log control, 1-5
 in reporting accounting information, 8-2
 in retrieving file sets, 5-41
 in storing file sets, 5-36
 System process, 2-2
 System timer, 8-2, 8-17

T

Tape Mark, 6-56-6-58
 :TELL command, 8-8, B-4
 :TELLOP command, 8-9, B-4
 Temporary files, 5-16
 Terminal
 break function, subsystem, enabling/disabling,
 8-84, 8-85
 break function, system, enabling/disabling,
 8-83, 8-84
 changing characteristics of, 8-74-8-91
 connect-time, 1-5, 3-16, 3-22, 3-23, 8-2
 controllers for, 8-74
 IBM 2741 Selectric, special considerations for
 3-19, 8-75, 8-76, 8-78
 8-79
 input echo facility, enabling/disabling, 8-81-8-83
 input timer, enabling/disabling, 8-88, 8-89
 input timer, reading, 8-89
 line-termination characters for input, 8-90, 8-91
 parity-checking, enabling/disabling, 8-85
 reading tapes without X-OFF control from, 8-87,
 8-88
 special keys, 8-76, 8-78
 specification of type :JOB command, 3-12
 speed, changing of, 8-78-8-81
 tape-mode option, enabling/disabling, 8-86, 8-87
 time-out interval for, 6-56-6-58
 types supported, 8-74, 8-75
 TERMINATE intrinsic, 8-47, 11-14, C-78
 Throughput, 1-4
 Time, printing of, 8-2, 8-18
 TIMER intrinsic, 8-18, C-79
 Top-of-stack register, 2-16, 2-17, 2-18, 2-19, 2-20
 TR-register, 2-18
 TRACE/3000, 4-23
 Traps, 8-30-8-43
 arithmetic, 8-30-8-33, 8-37, 8-38
 arithmetic, hardware, 8-31, 8-37, 8-38
 arithmetic, software, 8-30, 8-32
 CONTROL-Y traps, 8-35, 8-36, 8-43
 library, 8-30, 8-33, 8-34, 8-39, 8-40
 procedure, 8-30, 8-36-8-43
 system, 8-30, 8-34, 8-40, 8-41, 8-42

U

UCOP, 2-2, 2-5, 2-10, 2-23
Uncallable intrinsics, 2-8, 14-1
Undefined-length records, 5-15, 6-15
UNLOADPROC intrinsic, 7-29, C-80
UNLOCKGLORIN intrinsic, 9-4, C-81
UNLOCKLOCRIN intrinsic, 9-8, C-82
Updating files, 6-45
-USE command, 7-5, B-6
User
 attributes, 2-27, 8-19
 definition of, 2-24
 name, 2-24, 3-11, 3-19-3-21
 password, 2-24, 3-11
User Attributes, 2-27, 8-19
User Controller Process, 2-2, 2-5, 2-10, 2-23
User Pre-Defined Files, 4-1, 4-2, 5-5
User-managed area of stack, 2-16, 2-17, 4-13, 4-18,
 4-22, 7-14, 8-44, 8-45,
 11-9
User process, 2-2
User Subprogram Library
 adding new segment names to, 7-8
 changing directory/information block size on, 8-72
 changing size of, 8-72, 8-73
 copying RBM's to, 7-9
 creation of, 7-3
 definition of, 2-13, 7-1
 deleting RBM's on, 7-7
 designating for management, 7-3, 7-4
 entry-points in, activation of, 7-5
 entry-points in, deactivation of, 7-6
 in compilation, 2-13, 4-7, 7-1
 in preparation, 2-13, 4-6, 4-13, 4-17, 7-1
 initializing buffers for, 8-71
 listing RBM's on, 7-10-7-12
 management of, 7-1-7-15
 modification of, 7-1
 preparation into a program file, 7-12
USL
 adding new segment names to, 7-8
 changing directory/information block size on, 8-72
 changing size of, 8-72, 8-73
 copying RBM's to, 7-9
 creation of, 7-3
 definition of, 2-13, 7-1
 deleting RBM's on, 7-7
 designating for management, 7-3, 7-4
 entry points in, activation of, 7-5
 entry points in, deactivation of, 7-6
 in compilation, 2-13, 4-7, 7-1

in preparation, 2-13, 4-6, 4-13, 4-17, 7-1
initializing buffers for, 8-71
listing RBM's on, 7-10-7-12
management of, 7-1-7-15
modification of, 7-1
preparation into a program file, 7-12
-USL command, 7-3, B-6

V

Variable-length records, 1-3, 5-15, 6-5
Virtual Memory
 definition of, 1-4
 use of, 2-2

W

WHO intrinsic, 8-19, C-83
Write-access, 5-15, 6-25
Writing on files, 6-37-6-41

X

XARITRAP intrinsic, 8-31, 8-32, 8-39, C-84
XCONTRAP intrinsic, 8-36, C-85
XLIBTRAP intrinsic, 8-33, 8-34, C-86
XSYSTRAP intrinsic, 8-34, C-87
X-OFF control character, 8-87, 8-88

Z

Z-DL area of stack, 2-16, 2-17, 4-13, 4-18, 4-21, 7-14,
 8-44-8-46, 11-10
Z-Q area of stack, 2-17, 4-13, 4-18, 4-22, 7-14, 11-10
Z-register, 2-16-2-20
ZSIZE intrinsic, 8-45, 8-46, C-88

Index of Commands

A

:ABORT, 3-23, B-1
-ADDRL, 7-19, B-5
-ADDSL, 7-24, B-5
:ALTSEC, 5-51, B-1
-AUXUSL, 7-9, B-5

B

:BASIC, 4-5, B-1
:BASICGO, 4-11, B-1
:BASICOMP, 4-7, B-1
:BASICPREP, 4-9, B-1
:BUILD, 5-30, B-2
-BUILDRL, 7-18, B-5
-BUILDSL, 7-23, B-5
-BUILDUUSL, 7-3, B-5
:BYE, 3-22, 3-23, B-2

C

-CEASE, 7-6, B-5
:COBOL, 4-7, 4-8, B-2
:COBOLGO, 4-11, B-2
COBOLPREP, 4-9, B-2
:COMMENT, 8-6, B-2
:CONTINUE, 3-9, 3-25, 8-51, B-2
-COPY, 7-9, B-5

D

:DATA, 3-25, 3-26, 6-5, B-2

E

:EDITOR, 4-22, B-2
:EOD, 3-16, 3-24, B-2
:EOJ, 3-16, B-2
-EXIT, 7-2, B-5

F

:FILE, 5-11-5-26, 6-11, B-3
:FORTGO, 4-11, B-3
:FORTPREP, 4-9, 4-10, B-3
:FORTRAN, 4-7, 4-8, B-3
:FREERIN, 9-6, B-3

G

:GETRIN, 9-2, B-3

H

:HELLO, 3-19-3-22, B-3
-HIDE, 7-27, B-5

J

:JOB, 3-10-3-15, B-3

L

:LISTF, 5-32, B-3
-LISTRL, 7-20, B-5
-LISTSL, 7-24, B-5
-LISTUSL, 7-10, B-5

N

-NEWSEG, 7-8, B-5

P

:PREP, 4-12, 4-13, B-3
-PREPARE, 7-12, B-5
:PREPRUN, 4-17, B-4
:PTAPE, 8-87, B-4

:PURGE, 5-32, B-4
-PURGERBM, 7-7, B-5
-PURGERL, 7-19, B-6
-PURGESL, 7-24, B-6

R

:RELEASE, 5-53, B-4
:RENAME, 5-44, B-4
:REPORT, 8-2, B-4
:RESET, 5-11, 5-29, B-4
:RESETDUMP, 8-65, B-3
:RESTORE, 5-40, B-4
:RESUME, 8-46, B-4
-REVEAL, 7-27, B-6
:RJE, 4-25, B-5
-RL, 7-19, B-6
:RPG, 4-7, B-5
:RPGGO, 4-11, B-5
:RPGPREP, 4-9, B-5
:RUN, 4-21, B-5

S

:SAVE, 5-31, B-5
:SECURE, 5-53, B-5

:SEGMENTER, 4-24, 7-2, B-5
:SETDUMP, 8-64, B-5
:SETMSG, 8-9, B-5
:SHOWDEV, 8-4, B-5
:SHOWIN, 8-5, B-5
:SHOWJOB, 8-2, B-6
:SHOWOUT, 8-5, B-6
:SHOWTIME, 8-2, B-6
-SL, 7-23, B-6
:SPEED, 8-78, 8-79, B-6
:SPL, 4-7, 4-8, B-6
:SPLGO, 4-11, B-6
:SPLPREP, 4-9, B-6
:STAR, 4-23, B-6
:STORE, 5-35-5-40, B-6, F-1, F-2
:STREAM, 6-10, B-6

T

:TELL, 8-8, B-6
:TELLOP, 8-9, B-6

U

-USE, 7-5, B-6
-USL, 7-3, B-6

Index of Intrinsic

A

ACTIVATE, 11-12, C-2
ADJUSTUSLF, 8-72, 8-73, C-3
ALTDSEG, 12-7, C-4
ARITRAP, 8-31, C-5
ASCII, 8-11, C-6

B

BINARY, 8-10, C-7

C

CALENDAR, 8-18, C-10
CAUSEBREAK, 8-46, 8-83, C-8
CHECKDEV, 8-70, C-9
CHRONOS, 8-18, C-10.1
CLOCK, 8-18, C-10.2
COMMAND, 8-29, C-11
CREATE, 11-7, 14-9, C-12
CTranslate, 8-13, C-12.1

D

DASCH, 8-13, C-13
DBINARY, 8-11, C-14
DEBUG, 8-48-8-67, C-15
DLSIZE, 8-44-8-45, C-16
DMOVIN, 12-4, 12-5, C-17
DMOVOUT, 12-5, 12-6, C-18

E

EXPANDUSLF, 8-72, 8-73, C-19

F

FATHER, 11-25, C-20
FCHECK, 6-52-6-55, 8-85, C-21
FCLOSE, 6-2, 6-30-6-32, C-22
FCONTROL, 6-56, 8-80-8-91, C-23
FGETINFO, 6-48-6-51, C-24

FLOCK, 6-60, 9-9, 9-10, 13-1, C-25
FOPEN, 6-1, 6-17-6-30, C-26
FPOINT, 6-47, C-27
FREAD, 6-32, 6-33, C-28
FREADDIR, 6-34, C-29
FREADLABEL, 6-43, C-30
FREADSEEK, 6-36, C-31
FREEDSEG, 12-3, 12-4, C-32
FREELOCIN, 9-9, C-33
FRELATE, 6-61, 6-62, C-43
FRENAME, 6-60, 6-61, C-35
FSETMODE, 6-58-6-60, C-36
FSPACE, 6-46, C-37
FUNLOCK, 6-60, 9-11, C-38
FUPDATE, 6-45, C-39
FWRITE, 6-37-6-41, C-40
FWRITEDIR, 6-41, C-41
FWRITELABEL, 6-44, C-42

G

GETDSEG, 12-1-12-3, 14-5, 14-6, C-43
GETJCW, 8-69, C-44
GETLOCIN, 9-6, C-45
GETORIGIN, 11-25, C-46
GETPRIORITY, 11-23, 11-24, 11-26, 14-9, C-47
GETPRIVMODE, 14-3, C-48
GETPROCID, 11-26, C-49
GETPROCINFO, 11-28, C-50
GETUSERMODE, 14-4, C-51

I

INITUSLF, 8-71, 8-73, C-52

K

KILL, 11-16, C-53

L

LOADPROC, 7-28, C-54
LOCKGLORIN, 9-3, 13-1, C-55
LOCKLOCRIN, 9-7, C-56

M

MAIL, 11-17, C-57
MYCOMMAND, 8-25, C-58

P

■ PAUSE intrinsic, 8-18A, C-58.1
PRINT, 8-15, C-59
■ PRINTOP, 8-16, C-60
PRINTOPREPLY, 8-16, C-61
PROCTIME, 8-22, C-62
PTAPE, 8-88

Q

QUIT, 8-47, C-64
QUITPROG, 8-48, C-65

R

READ, 8-13, C-66
READX, 8-15, C-66
RECEIVEMAIL, 11-20-11-22, C-68
RESETCONTROL, 8-36, C-69
RESETDUMP, 8-66, C-70

S

■ SEARCH, 8-22-8-24, 8-26, C-71
SENDMAIL, 11-18, C-72
SETDUMP, 8-65, C-73
SETJCW, 8-69, C-74
STACKDUMP, 8-59, C-75
SUSPEND, 11-13, C-76
SWITCHDB, 14-5, C-77

T

TERMINATE, 8-47, 11-14, C-78
TIMER, 8-18, C-79

U

UNLOADPROC, 7-29, C-80
UNLOCKGLORIN, 9-4, C-81
UNLOCKLOCRIN, 9-8, C-82

W

WHO, 8-19, C-83

X

XARITRAP, 8-31, 8-32, 8-39, C-84
XCONTRAP, 8-36, C-85
XLIBTRAP, 8-33, 8-34, C-86
XSYSTRAP, 8-34, C-87

Z

ZSIZE, 8-45, 8-46, C-88

READER COMMENT SHEET

**MPE/3000 Operating System
Reference Manual**

32000-90002

January 1975

We welcome your evaluation of this manual. Your comments and suggestions help us improve our publications. Please use additional pages if necessary.

Is this manual technically accurate?

Is this manual complete?

Is this manual easy to read and use?

Other comments?

FROM:

Name _____

Company _____

Address _____

FOLD

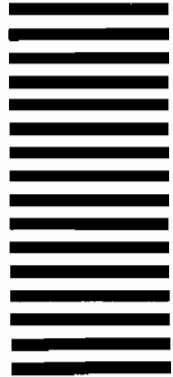
FOLD

BUSINESS REPLY MAIL

No Postage Necessary if Mailed in the United States Postage will be paid by

Manager, Technical Publications
Hewlett-Packard
Data Systems Division
11000 Wolfe Road
Cupertino, California 95014

FIRST CLASS
PERMIT NO. 141
CUPERTINO
CALIFORNIA



FOLD

FOLD