MPE IIA

External Reference Specifications

Project Number: 4752-3065

Location Code : 47-7600
Date : April, 1977

Alan Hewer
Ed Basart
Ron Hoyt
Dan Lundgren
Neal Mack
Bob Olson
Bob Vannucci

USER LOGGING

IDENTIFICATION

Name

MPE IIB USER LOGGING

.1.2 Abstract

MPE IIB User Logging provides a flexible new logging capability for MPE IIB users. The facility supports both system and private level logging. System level logging routes all log records to a common disc or tape file. Private logging dedicates a disc or tape file to a particular user.

.2 DESIGN CONSIDERATIONS

.2.1 Hardware Environment

USER LOGGING is an integral part of MPE IIB. It will run with any of the supported software on the system. Logging over DS lines has not been considered in the project.

.2.2 Software Environment

USER LOGGING will run on any hardware that runs MPE IIB. If logging to tape is desired, the hardware configuation must have a tape drive.

.3 DESCRIPTION

.3.1 Overview of Operation

MPE IIB user logging provides an important new facility for MPE IIB users. It allows users and subsystems to journalize additions and modifications to MPE IIB and subsystems files, and provides a mechanism whereby the journal entries can be used to recover these files in the event that it becomes necessary.

MPE IIB logging provides the user with two unique types of logging. For the average user, there is the system logging facility which takes all log records and writes them to one central logging file. The logging identifier as described in this document is used to distinguish ownership of the individual log records for the recover utility. This type of logging has the advantage of making both tape and disc files sharable resources. Thus the facility improves the useability of an important system resource while logging.

The other type of logging provided by the facility is private logging. This method can be used when it is necessary for the application to maintain it's own transaction records for

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

reasons of recovery, auditing, or security. With this type of logging, all predetermined logging identifiers that have been assigned to a private logging process are separated by the logging facility and written to a private disc or tape file.

Both private and system logging operate in the same manner so the description that follows applies to both.

The MPE IIB logging facility consist of a logging buffer file, a logging process, a logging data segment, and an optional logging tape file for each logging process. The logging buffer file has two functions in the system. First, if the operator requests logging to tape, the logging buffer file buffers I/O between the user's process and the logging tape file. This will reduce the amount of overhead due to I/O to the logging tape. It also provides for more "even" response to interactive users who won't "feel" the I/O to tape taking place. If the operator specifies logging to disc, the logging buffer assumes the role of the logging file. When this occurs, all records received by the logging process will remain in the logging buffer. The file will expand, within limits, as the need for more space occurs.

The logging process acts as the interface between the logging buffer file and the logging tape file when logging to tape is requested by the operator. This process runs independent of the user's process and thus the logging process can be writing records to tape at the same time that users are adding additional records to the logging buffer. Again, this aids in making the users response from the logging facility quick and even. When logging to disc is specified, the logging process function changes. In this mode of operation, the process insures that enough disc space is available so that logging can continue. It does this by opening additional logging files when the current file reaches a predetermined point. Thus, when logging to disc, users should rarely be impeded while more disc space is being made available.

The logging data segment is divided into two areas. First there is the buffer area which is used to buffer I/O between user's processes and the logging buffer. Secondly, there is the communications area in which users and the logging process send messages. These messages generally pertain to the status of the users process or state of the logging process.

Users accessing the log buffer file do so by placing their records in the buffer area of the logging data segment via intrinsics. The process that fills the logging buffer area of the data segment writes it's contents to the log buffer file. It then checks the communications area of the logging data segment to insure that the logging process is active. If the logging process is sleep, it activates the logging process.

TABLE OF CONTENTS

4. PRIVATE VOLUMES

5. USER LOGGING

6. SERIAL DISC INTERFACE

6.1 IDENTIFICATION
- 6.1.1 Name
- 6.1.2 Abstract

6.2 DESIGN CONSIDERATIONS
- 6.2.1 Hardware Environment
- 6.2.2 Software Environment
- 6.2.3 Performance / Customer Satisfaction

6.3 DESCRIPTION / USER INTERFACE
- 6.3.1 Comparison to Magnetic Tape
- 6.3.2 Privileged Mode Users Only

6.4 OPERATOR INTERFACE
- 6.4.1 Operator Control
- 6.4.2 Use as a Cold Load Medium

6.5 INTERNAL CONSIDERATIONS
- 6.5.1 Auto Recognition
- 6.5.2 Declaration of Serial Disc Class
- 6.5.3 References to Serial Disc by Class or Ldev
- 6.5.4 Disc Label
- 6.5.5 Defective Tracks Table
- 6.5.6 Data Storage / Record Format
- 6.5.7 End of Tape
- 6.5.8 Device Directory (XDD) Management
- 6.5.9 Track Zero Table

7. STORE/RESTORE ENHANCEMENTS

7.1 DISCUSSION

7.2 STORE MEDIA
- 7.2.1 Current Media
- 7.2.2 Future Media
- 7.2.3 Observation

7.3 STORE DATA STRUCTURE
- 7.3.1 Definition of Symbols and Elements
- 7.3.2 Old STORE Data Structure
- 7.3.3 New STORE Data Structure

7.4 RESTORE STRATEGY

7.5 INTERNAL INTERFACE CONSIDERATIONS
- 7.5.1 IMAGE

9. DISC CONDENSE FACILITY

9.1 DISCUSSION

9.2 CAPABILITIES

9.3 OPERATION

10. BACKUP AND RECOVERY

2. MPE IIB OVERVIEW

2.1 DISCUSSION

Due to the independent nature of the topics being presented in this document, the requirement exists to summarize the total impact on the external characteristics of MPE. The intent of the overview is to address the following categories of MPE facilities:

- MPE Commands
- MPE Intrinsics
- Operator Commands
- Subsystems / Utilities

and within each category indicate either a new facility or a modification to a currently existing facility. Further, as items are listed, references are made to the relevent topics in the document which are responsible for and carry a detailed description of the enhancement.
In some cases, a reference may be given in order to indicate the indirect benefits of a functional enhancement and not an actual change to the syntax of the facility.

ABBREVIATIONS

The following abbreviations are used to refer to topics and the corresponding sections within the document:

ABBREVIATION	TOPIC	SECTION #
UNCL	Unified Command Language	3
PV	Private Volumes	4
UserLog	User logging	5
SerDisc	Serial Disc Interface	6
St/Rest	Store/Restore Enhancements	7
GenName	Generic Names	8
DiscCon	Disc Condense Facility	9
BackRec	Backup and Recovery	10

MPE COMMANDS

.1 Modified MPE Commands

COMMAND	REFERENCES			
STORE	St/Rest	GenName	BackRec	PV
RESTORE	St/Rest	GenName	BackRec	PV
LISTF	GenName	PV		
LISTACCT	GenName			
LISTGROUP	GenName			
LISTUSER	GenName			
REPORT	GenName			
SHOWDEV	SerDisc			
SHOWIN	SerDisc			
SHOWOUT	SerDisc			
BUILD	PV			
RENAME	PV			
PURGE	PV			
SAVE	PV			
ALTSEC	PV			
RELEASE	PV			
SECURE	PV			
NEWACCT	PV	UserLog		
ALTACCT	PV	UserLog		
PURGEACCT	PV			
NEWGROUP	PV			
ALTGROUP	PV			
PURGEGROUP	PV			
LISTUSER	PV			
LISTGROUP	PV			
LISTACCT	PV			
GETLOG	UserLog			
RELLOG	UserLog			
LISTLOGID	UserLog			
ALTLOGID	UserLog			
SHOWLOGSTATUS	UserLog			
NEWUSER	UserLog			
ALTUSER	UserLog			

.2 New MPE Commands

COMMAND	REFERENCES
HELP	UNCL
?	UNCL
REDO	UNCL
SAVEJCW	UNCL
SETJCW	UNCL
IF THEN	UNCL
ELSE	UNCL
ENDIF	UNCL
SETCATALOG	UNCL
CHECKPOINT	UNCL

MPE INTRINSICS

Modified MPE Intrinsics

INTRINSIC	REFERENCES
FOPEN	PV

New MPE Intrinsics

INTRINSIC	REFERENCES
PROGTERM	UNCL
OPENLOG	UserLog
WRITELOG	UserLog
CLOSELOG	UserLog
LOGINFO	UserLog

NEWVSET PV

OPERATOR COMMANDS

Modified Operator Commands

COMMAND	REFERENCES
SHOWDEV	SerDisc
SHOWIN	SerDisc
SHOWOUT	SerDisc
UP	PV
DOWN	PV
REPLY	PV SerDisc

New Operator Commands

COMMAND	REFERENCES
LISTV	PV
DSTATUS	PV
MOUNT	PV
DISMOUNT	PV
VMOUNT	PV
LOG	UserLog

SUBSYSTEMS / UTILITIES

.1 Modified Subsystems / Utilities

FACILITY	REFERENCES			
INITIAL	SerDisc	BackRec		
SYSDUMP	SerDisc	GenName	BackRec	UNCL

.2 New Subsystems / Utilities

FACILITY	REFERENCES		
VINIT	DiscCon	PV	BackRec
RECOVER	UserLog		
MAKECAT	UNCL		

UNCL

1 IDENTIFICATION

1.1 Name

UNCL (Unified Command Language)

1.2 Abstract

The proposed command language retains the existing command set intact, including the existing syntax and semantics. The extensions occur in three major areas. First, the interpreter should become significantly friendlier through the provision of a comprehensive, context sensitive HELP facility which includes an improved error handling scheme. A new command is proposed to implement the HELP facility. A REDO Command for correcting incorrect commands is proposed. Secondly, the language should become more useful in the design of jobs through the addition of a set of job flow commands. This set of commands includes a Block IF statement. Third, a means whereby the user may define his own commands is proposed. This should be of value to both inexperienced and experienced users.

1.3 Table of Abbreviations

CL - Command Language
CLI - Command Language Interpreter
CI - Command Interpreter
UNCL - sometimes refers to the project and sometimes refers to the new CI as modified by the UNCL project.
UDC - User Defined Command
JCW - Job Control Word, a 16 bit logical used for intrajob communication.

2. DESIGN CONSIDERATIONS

2.1 Hardware Environment

Any SERIES II or successor capable of running MPE IIB

2.2 Software Environment

MPE IIB. UNCL will be developed on the SERIES II software base

2.3 Performance

The performance of the CI as modified by this project should not be measured in terms of the code size or speed of execution. Such problems will be dealt with on an ad hoc basis as they appear. The success of this project really should be measured in terms of

improved user satisfaction with the system's ease of use, capability and flexibility.

This improved user satisfaction will be achieved by careful attention to the principles derived from four basic perceptions of the user universe:

a) The typical user wants to solve non-computer problems with the least amount of incidental nonsense. Therefore the system dependent details should be as painless and self explanatory as possible.

b) The CI should be the USER'S tool for describing a task to the system. Therefore the externals of the CI should conform to the processes and modes of thought of the user. That is, it should be natural to use, in some sense.

c) Most users have simple problems, programs and jobs. Therefore simplicity, friendliness and usability are primary concerns.

d) Some users have complex problems and correspondingly complex programs and systems of programs. Furthermore these users tend to be the heaviest users of the systems and thus tend to be most affected by the cost/effectiveness ratio. Therefore some attention must be paid to the problems of this class of users. Simple, cheap tools may greatly increase productivity.

e essential idea behind the user aids is that the command
terpreter itself should have an active role in helping the user
nerate correct command statements. In the Design Objectives
is idea was translated into two basic UNCL design goals:

1) Make the level of knowledge required to use the command language as low as possible.

2) Make error detection and correction as painless and understandable as possible.

n more concrete terms, it appears that there are four general situations
n which the user requires assistance.

1) He wants to do something, but he doesn't know what command or commands to use. For example, he wants to do a file equate.

2) He knows which command he needs, but he doesn't remember the syntax exactly. For example, perhaps he doesn't remember the order of the sub-parameters in the DISC parameter.

3) He remembers the syntax, but he is not sure of the semantic details. For example, he may not be sure if the file size sub-parameter is expressed in logical records or physical records.

4) An error has occurred and he doesn't understand what the problem is or how to correct it. For example, he may have attempted to allocate an illegally large number of extents for his disc file.

The required assistance can be provided
in two ways. First, the user may explicitly request context insensitive
information through the HELP command.
Alternatively, the user can be provided assistance
involuntarily when he makes an error. In either case, the emphasis
is on providing information specific to whichever of
those four situations seems appropriate.

On-Line Help Facility

The heart of the Help facility is a tree organized set of
messages. Its primary function is to provide the user with graduated,
increasingly detailed answers to questions about some
aspect of the command set. A basic assumption for the Help facility
is that the on-line user typically is somewhat familiar with the command
language. He may, however, have forgotten some detail of syntax or semantics,
or even the name of the particular command required. Thus the

ategy will be to provide short, concise pieces of information which
an be quickly located in a predictable manner by the user.
The set of messages will be maintained
by publications personnel with the assistance of the Lab.

is set of messages will be easily
ditiable and extendable by both H.P. personnel and by our customers.

The first way of entering the Help message tree is through the HELP command. The information provided by this command is not context sensitive. The syntax of HELP is

```
:HELP [HELP                          ]
      [<tablecontents>               ]
      [<command> [,<keyword>]]
      [ALL                           ]
```

:HELP with parameter(s) causes HELP to find information, display and return to the CI, which prompts (:). Break and control Y abort the command. Control S causes output and control Q resumes output. :HELP <carriage return> causes HELP to display a menu and then prompt (>) for further input.
HELP is then in "subsystem" mode. While in subsystem mode the control functions described above have a slightly different definition.
Break aborts the command,
control Y flushes output up the the next information node,
<carriage return> causes HELP to display the next node of information (<carriage return> is like page turning), and control S stops output while control Q resumes. Syntax for "subsystem" mode:

```
>[HELP                               ]
 [<tablecontents> [,<keyword>]]
 [<keyword>                          ]
```

For both modes:

<tablecontents>	Gives a list of commands by groups. Group names are: SESSIONS JOBS PROGRAMS FILES MANAGE UTILITY
<command>	Gives syntax description of the command, followed by a keyword list. May be any MPE command.
<command>, <keyword>	Gives detail command description for <command>. Keyword can be any keyword from the keyword list. Typing <command>, ALL gives complete command description.

:HELP ALL displays menu, each command group and the list of commands found in that group.

In "subsystem" mode >ALL displays any remaining information for the current <command>. If no <command> has been entered, HELP displays the same information as :HELP ALL. Typing <keyword> (from keyword list) displays information about <keyword>. This gives the most detailed information mode from the HELP manual. >HELP in subsystem mode is the same as :HELP HELP.

The simplest case is the use of :HELP with no parameters. This causes entry at the root node of the tree. A typical response to this command might be

```
Information is available on the following classes of commands:
  Running Sessions
  Running Jobs
  Managing Files
  Running Subsystems and Programs
  System Management, Status, and Accounting
  Utility functions

For more information, enter a KEYWORD.  You can also enter any command
name as a keyword.  Enter 'help' for information on help.  Enter 'exit'
to leave help.
```

The second case is where <keyword> is specified. This gains the user entry into the tree at the point specified by <keyword>. For example,

```
:HELP SESSIONS
```

might result in a brief description of the commands in that class. Similarly,

```
:HELP ABORT
```

might result in

```
ABORTS EXECUTION OF CURRENT PROGRAM.
  :ABORT
KEYWORDS:  USE, OPERATION, EXAMPLE
```

A more complicated example is

```
:HELP SPLPREP
```

which might result in

```
COMPILES AND PREPARES AN SPL PROGRAM
   :SPLPREP [TEXTFILE][,[PROGFILE][,[LISTFILE][,[MASTERFILE]
                [,NEWFILE]]]
KEYWORDS:  PARAMETERS, USE, OPERATION, EXAMPLE
```

Finally,

```
:HELP SPLPREP,EXAMPLE
```

might result in the following reply:

> To compile and prepare an SPL source program input from a source file named SFILE into a program file named MYPROG, with the resulting listing generated on the job/session list device (and no master or new file) enter the following command:
>
> :SPLPREP.SFILE,MYPROG

There are several things to be noted about these examples. First, the terminal node of the tree has a manual reference instead of a list of further keywords. Second, the selection of keywords and sub-keywords is quite regular. After a few times, the user should be able to reach right into the tree for the specific node that he wants, without having to chase down the entire tree each time. For example, it should be obvious from the above that one could specify

:HELP STREAM,PARAMETERS

to find out the details of the parameters for the STREAM command. Third, it should be noted that the keywords and the text of these examples were taken directly from the MPE COMMANDS manual. Chapter 2 of that manual provides exactly the sort of information which should be available from the Help facility. By copying it almost verbatim, we achieve consistency between the two, besides saving development time.

There are two additional minor details. First, in order that there be as few errors as possible in using the HELP command, any delimiter, including blanks, will be allowed. The second detail is that ALL will be allowed as a keyword at any level. The effect of ALL will be to have the entire sub-tree displayed. As usual with the other CI commands, the break key may be used to stop the printing of text.

.2 Improved Error Handling Capabilities

The second major class of user aids proposed for the CLI are improvements in the way it handles user errors. The basic strategy is as follows:

1. A precise, concise indication of an error should be given to the user.
2. If appropriate, the user should be given the opportunity to edit the offending command rather than being required to re-enter the line.

In what follows, these two principles are translated directly into functional capabilities for the command language.

Error Messages

A common problem with error messages is that they tend to indicate what the problem was from the subsystem's point of view, not the user's. This is because statement parsers frequently are designed just to filter out incorrect statements. In such cases, the error message is used simply to indicate to the user that the statement was not accepted. An extreme case is the *INVALID* given the console operator upon any incorrect command. If our CLI is to help the user generate correct commands, if at all possible it must generate messages which help the user understand what is correct, not just what is incorrect. To be specific, where appropriate, the position in the command where the error was detected should be indicated with a caret and a useful message be displayed. The message should indicate in specific terms what the error was and suggest corrective action.

Example:

```
Current CI -
  :FILE A;REC=,,G;DISC=10000,16,1;SAVE
  ERR 22,5
  ILLEGAL PARAMETER

Proposed CI -
  :FILE A;REC=,,G;DISC=10000,16,1;SAVE

  EXPECTED RECORD FORMAT OF F, V OR U.  (CIERR 273)
```

3.2.2 Editing of Erroneous Commands

The third error handling improvement proposed for the CLI is a mechanism whereby the user may correct certain kinds of errors by editing the incorrect command rather than re-entering the whole command. In general this would include any command where the CLI itself was able to detect the error. Included are such errors as missing or incorrect punctuation and parameters, spelling errors and so forth. Excluded would be errors in the logon command, most subsystem detected errors and similar problems.

The command editing mode would be available through the agency of a REDO command. This command would cause the CLI to go into an Editor-like modify mode, operating on the immediately preceeding command string. Once the editing was complete the CLI should attempt to re-execute the edited command. If the BREAK key was hit, the edit mode would be exited & the command being edited would be discarded. The user may REDO the preceeding command even if no error was detected. The edit Functions available are D(elete), I(Insert), R(eplace) and U(ndo). Delete, Insert and Replace are defined as in the Editor modify command. The Undo

function will back up by one edit function. Two consecutive Undo's will back the edit up to the original form of the command. If the edit request is not one of D, I, R or U then it is executed as a Replace operation.

Example 1:

```
:FILE A;REC=,,G;DISC=10000,16,1;SAVE   cr   ##cr => carriage return

EXPECTED RECORD FORMAT OF F, V OR U.  (CIERR 273)
:REDO   cr                          ##request to edit command string
FILE A;REC=,,G;DISC=10000,16,1;SAVE  ##retyped to prompt user
          RF   cr                   ##replace the "G" with "F"
FILE A;REC=,,F;DISC=10000,16,1;SAVE   ##edited command retyped by CLI
cr                                  ##user satisfied - execute command
```

Example 2:

```
:FILE A;REC=,,G;DISC=10000,16,1;SAVE  cr     ##same example

EXPECTED RECORD FORMAT OF F, V OR U.  (CIERR 273)
:LISTF     ##user decided not to fix up the command
```

The essential idea behind the error handling improvements, and indeed the whole user aids part of the UNCL project, is that the CLI should assume a role in the preventing, diagnosing and correcting of user errors. It is further believed that, for most users, the command language is simply a means to an end, not the end itself. Thus the CLI should be as friendly and supportive as possible. The various user aids described above are an attempt to reach this goal. Specifically, they seem to lower the level of knowledge required to use the command language and ease error handling. This package should substantially improve the usability of our command language.

FLOW OF CONTROL

The primary design goal for the job step sequencing commands is to provide at least a primitive means for dynamically altering the order of execution of the various elements of a job. Such control functions are useful primarily in the creation of production batch jobs, since the components of such jobs frequently may fail in less than fatal ways. Conditional execution functions allow the job designer to execute subsequent job steps conditionally on the results of the previous job steps.

Effective job control relies on four major elements:

1) A means of passing status information between the CI and programs.
2) A means whereby such information can be examined in the CI.
3) A means of altering the path of execution based upon an examination of the passed information.
4) A means whereby job step status information may be preserved for later job steps to examine.

To meet these requirements this project proposes an enhanced definition of the Job Control word (JCW), a command to allow saving and setting the JCW(S), and a set of commands to implement a Block IF construct.

.1 Job Control Word (JCW)

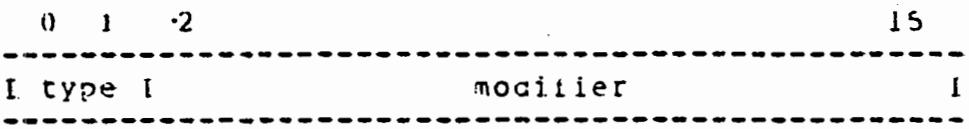
The JCW is a sixteen bit logical which is accessible to all processes in a job through intrinsics for intrajob communication. At present the command interpreter assigns a special meaning to only bit 0 of that word. If on return from execution of a program, bit 0 is set, the CI assumes that a fatal error of some sort has occurred. If in a session, the rest of the current command is flushed, or if in a job (batch) the rest of the job may be flushed. This sort of one bit go/no go control flag is not sufficient for effective, fault tolerant job control.

It appears that the control information necessary for job step sequencing can be partitioned into four general classes. In a crude sort of way, these classes will be defined as follows:

OK - The previous job step executed correctly.
WARN - Something non-fatal but unusual was discovered in the previous job step.
FATAL - Something severely wrong was discovered in the previous job step. Set by user (program).
SYSTEM - The system aborted the previous job step. This

includes such things as bounds violations and similar failures outside the control of the user (program).

Bits 0 and 1 of the JCW will be used as a type field to indicate to which of these classes the modifying information in the other fourteen bits belongs.



TYPE: 00 => OK
01 => WARN
10 => FATAL
11 => SYSTEM

There are three important points to be made about this redefinition of the JCW. First, it is almost completely upward compatible with the current definiton and use of the JCW. In particular, bit 0 will still be a fatal error flag. The second important point is that the modifier can take up to 16K different values, enough for almost any application. Finally, the definitions of the various classes are only suggestions to which HP developed software should adhere. There is nothing to prevent alternate definitions by users with special needs. (Although the features described below will make it much easier to follow these definitions than to use one's own definitions).

4.2 Programmatic Access to JCW

With one exception, the user program must explicitly set the JCW in order to change its value. On entry to the program, the JCW has the same value it had just before the program was started. The value of the JCW may be read at any time through the use of the existing GETJCW intrinsic. Similarly it may be set through the SETJCW intrinsic. If the user stops execution in a normal manner, such as a call to TERMINATE, the JCW will be left unaffected. If a user exits with the QUIT intrinsic, the type will be set to FATAL and the numeric parameter to QUIT will be put into the JCW as the modifier. The one exception to the requirement that the user explicitly set the JCW is when the program terminates due to some problem outside the control of the program, such as a bounds violation. In such cases the type will be set to SYSTEM and the modifier will indicate what the error was.

A compiler might use the JCW in the following way. If errors were detected, the type would be set to FATAL and the error count put into the modifier. If only warnings were detected, the compiler might set the WARN flag and indicate the number of warnings encountered. Otherwise it might set the JCW to zero to indicate that the compile was clean. Such a scheme would enable subsequent job steps to be executed based on such

fined information as the number of warnings encountered in the compile.

The current assignment of "SYSTEM" codes is as follows:

SYSTEM0 - The previous program was aborted by a request from the user external to the program. (i.e., :ABORT)
SYSTEM1 through SYSTEM1000 are for non-intrinsic errors
SYSTEM1 - integer overflow
SYSTEM2 - floating point overflow
SYSTEM3 - floating point underflow
SYSTEM4 - integer divide by zero
SYSTEM5 - floating point divide by zero
SYSTEM6 - privileged mode instruction abort
SYSTEM7 - unimplemented instruction abort
SYSTEM8 - extended precision floating point overflow
SYSTEM9 - extended precision floating point underflow
SYSTEM10 - extended precision floating point divide by zero
SYSTEM11 - decimal overflow
SYSTEM12 - invalid ASCII digit in decimal instruction
SYSTEM13 - invalid decimal digit
SYSTEM14 - invalid source word count for decimal instruction
SYSTEM15 - invalid decimal operand length
SYSTEM16 - decimal divide by zero
SYSTEM17 - SIT uncallable
SYSTEM18 - unused
SYSTEM19 - unused
SYSTEM20 - stack overflow
SYSTEM21 - unused
SYSTEM22 - bad stack marker
SYSTEM23 - memory address not in physical memory
SYSTEM24 - bounds violation
SYSTEM25 - non-responding module
SYSTEM26 - unused
SYSTEM27 - unused
SYSTEM28 - unused
SYSTEM29 - stack underflow
SYSTEM30 - CST violation
SYSTEM31 - SIT violation
SYSTEM32 to SYSTEM1000 unused, reserved for future expansion.

SYSTEM1001 to SYSTEM16383 are for intrinsic errors.
The modifier minus 1000 is the intrinsic number.
SYSTEM1001 - FOPEN
SYSTEM1002 - FREAD
SYSTEM1003 - FWRITE
SYSTEM1004 - FUPDATE
SYSTEM1005 - FSPACE
SYSTEM1006 - FPOINT
SYSTEM1007 - FREADDIR
SYSTEM1008 - FCLOSE
SYSTEM1010 - FCHECK
SYSTEM1011 - FGETINFO
SYSTEM1012 - FREADSEEK

SYSTEM1013 - FCONTROL
SYSTEM1014 - FSETMODE
SYSTEM1015 - FLOCK
SYSTEM1016 - FUNLOCK
SYSTEM1017 - FRENAME
SYSTEM1018 - FRELATE
SYSTEM1019 - FREADLABEL
SYSTEM1020 - FWRITELABEL
SYSTEM1021 - PRINTFILEINFO
SYSTEM1022 - IOWAIT
SYSTEM1030 - GETLOCRIN
SYSTEM1031 - FREELOCRIN
SYSTEM1032 - LOCKLOCRIN
SYSTEM1033 - UNLOCKLOCRIN
SYSTEM1034 - LOCKGLORIN
SYSTEM1035 - UNLOCKGLORIN
SYSTEM1040 - TIMER
SYSTEM1041 - CHRONOS
SYSTEM1042 - PROCTIME
SYSTEM1043 - CALENDAR
SYSTEM1044 - CLOCK
SYSTEM1045 - PAUSE
SYSTEM1050 - XARITRAP
SYSTEM1051 - ARITRAP
SYSTEM1052 - XLIBTRAP
SYSTEM1053 - XSYSTRAP
SYSTEM1054 - XCONTRAP
SYSTEM1055 - RESETCONTROL
SYSTEM1056 - CAUSEBREAK
SYSTEM1060 - TERMINATE
SYSTEM1061 - CTRANSLATE
SYSTEM1062 - BINARY
SYSTEM1063 - ASCII
SYSTEM1064 - READ
SYSTEM1065 - PRINT
SYSTEM1066 - PRINTOP
SYSTEM1067 - PRINTOREPLY
SYSTEM1068 - COMMAND
SYSTEM1069 - WHO
SYSTEM1070 - SEARCH
SYSTEM1071 - MYCOMMAND
SYSTEM1072 - SETJCW
SYSTEM1073 - GETJCW
SYSTEM1074 - DBINARY
SYSTEM1075 - DASCII
SYSTEM1076 - QUIT
SYSTEM1077 - STACKDUMP
SYSTEM1078 - SETDUMP
SYSTEM1079 - RESETDUMP
SYSTEM1080 - LOADPROC
SYSTEM1081 - UNLOADPROC
SYSTEM1082 - INITUSLF
SYSTEM1083 - ADJUSTSLF
SYSTEM1084 - EXPANDUSLF

SYSTEM1099 - DEBUG
SYSTEM1100 - CREATE
SYSTEM1102 - KILL
SYSTEM1103 - SUSPEND
SYSTEM1104 - ACTIVATE
SYSTEM1105 - GETORIGIN
SYSTEM1106 - MAIL
SYSTEM1107 - SENDMAIL
SYSTEM1108 - RECEIVEMAIL
SYSTEM1109 - FATHER
SYSTEM1110 - GETPROCINFO
SYSTEM1112 - GETPROCID
SYSTEM1120 - GETPRIORITY
SYSTEM1130 - GETDSEG
SYSTEM1131 - FREEDSEG
SYSTEM1132 - DMOVIN
SYSTEM1133 - DMOVEOUT
SYSTEM1134 - ALTDSEG
SYSTEM1135 - DLSIZE
SYSTEM1136 - ZSIZE
SYSTEM1139 - SWITCHDB
SYSTEM1191 - PTAPE
SYSTEM1192 to SYSTEM16383 are currently unassigned.

It is believed that this list is fairly complete. There are surely several intrinsics not listed, however.

4.3 Extension of the JCW Concept

The UNCL project has extended the JCW concept to allow the user to specify and manipulate other Job Control words besides the initially defined Job Control word "JCW". One potential problem with the existing JCW is that since HP software uses it for status information, the user cannot be absolutely sure that we will not modify it, destroying whatever information he may wish to pass. This extension essentially gives the user the ability to establish his own protocols which will be unaffected by our actions. In particular this may be a useful venicle for preserving status information between job steps.

A user defined Job Control word is a 16 bit logical which resides in an MPE managed table. This table, which also holds the system defined JCW, is shared by all processes in a job, thus assuring that any process can access any of the Job Control words. The name of a user defined JCW must start with an alpha character, consist of alphanumeric characters, and be between 1 and 255 characters long. It is defined and accessed through the new FINDJCW and PUTJCW intrinsics.

4.3.1 FINDJCW Intrinsic

FUNCTION: This intrinsic scans the JCW table for a given Job Control word, returning either its current value or an error indication.

DECLARATION:

```
PROCEDURE FINDJCW(JCWNAME, JCWVALUE, ERROR);
BYTE ARRAY JCWNAME;
LOGICAL JCWVALUE;
INTEGER ERROR;
```

INPUT PARAMETER:

JCWNAME — A byte array containing the name of the desired Job Control word. The name is terminated by any non-alphanumeric character.

OUTPUT PARAMETERS:

JCWVALUE — If no errors were encountered, this is the current value of the named JCW.

ERROR — The error indicator.
0 => No errors encountered.
1 => JCWNAME is greater than 255 characters long.
2 => The name does not start with an alphabetic character.
3 => The JCW named in JCWNAME does not exist.

CONDITION CODE: Unchanged.

4.3.2 PUTJCW Intrinsic

FUNCTION: This intrinsic scans the JCW table for a given Job Control word. If found, the Job Control word's value is updated to the value passed in the call. If it is not found, the name is inserted into the table, with JCWVALUE as its first value.

DECLARATION:

```
PROCEDURE PUTJCW(JCWNAME, JCWVALUE, ERROR);
BYTE ARRAY JCWNAME;
LOGICAL JCWVALUE;
INTEGER ERROR;
```

INPUT PARAMETERS:

JCWNAME — A byte array containing the name of the JCW to be set. The name is terminated by any non-alphanumeric character.

JCWVALUE — The value for the (perhaps newly defined) Job

Control Word.

OUTPUT PARAMETER:

ERROR The error indicator.
0 => No errors encountered.
1 => JCWNAME is greater than 255 characters long.
2 => The name does not start with an alphabetic character.
3 => The JCW table is out of space.

CONDITION CODE: Unchanged.

4.4 CI "Equates" for the JCW

For convenience in use and self-documentation, there are four classes of names which the CI will predefine as being equivalent to certain numeric values of the JCW. The meanings exactly parallel the subdivision of the JCW as described in section 4.1. To wit,

OK = 0
OK1 = OK+1 = 1
OK100 = OK+100 = 100 = %144

WARN = 16384 = %40000 (= OK16384)
WARN100 = WARN+100 = 16484 = %40144

FATAL = 32768 = %100000 (= WARN16384 = OK32768)
FATAL85 = FATAL+85 = 32853 = %100125

SYSTEM = 49152 = %140000
SYSTEM200 = SYSTEM+200 =%140310

OK < OK16383 < WARN < WARN16383 < FATAL < FATAL16383 < SYSTEM

These names will be reserved words whenever they are used in any of the commands described below for job control.

4.5 SETJCW Command

The SETJCW command essentially allows a user in the CI to call the PUTJCW intrinsic. Its syntax is

SETJCW <JCW name><any special character except %><value>

where <value> is one of

<octal number between 0 and %177777>
<decimal number between 0 and 65535>
<a JCW equate (see section 3.1.4)>
<another pre-existing JCW>

The value is transformed into its binary equivalent then assigned to <JCW name>. <JCW name> is created if it did not

exist before.

The SETJCW command provides the user with three capabilities. First, it allows the user to pass information into a program from the CI level. The existing mechanism for such information passing (PARM= in the RUN command) is virtually unusable from any language except SPL, since it requires explicit stack-conscious addressing. Since the JCW would be accessible with an intrinsic, the information would be available at any point in a program. Again, some extra intrinsics may be necessary to accommodate language specific problems. The second capability provided is a means whereby user defined commands and larger job steps could return condition codes. The third capability provided would be to allow jobs to abort themselves. Many jobs may wish to abort immediately upon receiving a bad error return from a program rather than going through some complicated IF statements. The way this would be achieved would be to set the FATAL flag in the system JCW. The SETJCW command would detect this flag and attempt to flush the job, following the usual rules.

In essence, the SETJCW command would allow the user to manipulate any JCW from the CI level of his task in a way complementary to the way his programs can manipulate it.

4.6 Block IF Commands

The block IF statement is used with the ENDIF statement and, optionally, the ELSE statement to control the execution sequence of a job. The syntax of the block IF statement is

```
:IF [(] <logical expression> [)] THEN
```

where <logical expression> is defined below.

An IF block consists of all the commands after the block IF statement up to, but not including, the next ELSE or ENDIF statement that has the same nesting level as the block IF statement. An ELSE block is similarly defined. Note that nesting to 15 levels is allowed. The ENDIF statement serves merely to delimit the IF block.

The execution of a block IF statement is as one would expect. The <logical expression> is evaluated, then, if true, the IF block is executed, else the ELSE block is executed, if one exists. IF blocks and ELSE blocks may be empty. The syntax of the ELSE statement is

```
:ELSE
```

and the syntax of the ENDIF statement is

```
:ENDIF
```

The <logical expression> consists of relationals or combinations of relationals. The allowed relational operators are >, <, >=, <=, =, <>. The allowed operands are JCW, any of the JCW values saved by name, any of the equates, or octal or decimal constants. Compound logical expressions can be formed using the AND and OR logical operators and nesting with parentheses as appropriate.

Example 1:

```
:SPL A,B,C,D
:IF (JCW < WARN10) THEN
:   PREP B,PROG
:   SAVE PROG
:ENDIF
```

In this example, we assume that the SPL compiler returned an error severity code in the system Job Control Word (JCW). The user expects a certain number of warnings and wishes to continue if that level is not exceeded. In this example the IF block would be executed if less than 10 warnings were detected. Since if any errors had occurred the JCW would have been set to a FATAL state, which is numerically larger than the largest WARN state, this test would also fail if any errors had been encountered.

Example 2:

```
                                    ##run first database update
                                    ##continue if ok so far
                                    ##save that return code
                                    ##run second database update
                                    ##continue if still ok
                                    ##save the second return code
                                    ##report current status of DB
                                    AND (UPDATE2JCW < 50) THEN

                                    (UPDATE2JCW = 1)) THEN

                                    update program failed -
                                    data base and report
```

Example 3: Batch Manufacturing Example

```
:CONTINUE
:RUN P108X1                    ##edit and verify transact
:CONTINUE
:RUN P108X2                    ##count valid transactions
:IF (JCW < FATAL) THEN         ##no fatal errors, schedul
:   IF (JCW < 5000) THEN       ##number of shipments to s
:      RUN P108X3              ##schedule low priority sh
:   ENDIF
:   RUN P108X4              ##schedule high priority ship
:ELSE
:   RUN P108X6                 ##produce error report (al
:ENDIF
:RUN P108X5                    ##produce final report
```

.7 SHOWJCW Command

The user may display the current state of any one or all J
Control Words with the command

:SHOWJCW [<JCW name>]

If no particular JCW is requested, all will be shown. For
example,

```
:SHOWJCW
JCW = USER
JCW1 = WARN0
```

.8 Avoiding "Abnormal Program Termination"

A subtle but significant change will be made to the :CONTI
command. It will still have the same effect of permitting
to continue even though the next command results in an err
The difference lies in its effect on the system defined JC
after a program or subsystem terminates. Currently it rese
the fatal error flag of the JCW (bit 0). Since the inform
that the program set the fatal bit could be useful at the
level, it appears that we should not destroy it by reseti
Accordingly, :CONTINUE will not affect the JCW, but will c
the CI to ignore the fatal bit for one command. The major
consequence to this change is that the user who wishes to
ignore certain fatal conditions would be well advised to r
the fatal error flag at the earliest opportunity. For exa

```
:CONTINUE
:RUN ABC
:SETJCW ABCJCW:=JCW
:SETJCW JCW := OK
:IF (ABCJCW=FATAL1) OR (ABCJCW=FATAL2) THEN
```

.9 Flow of Control Differences between the ERS's

1) Intrinsic PROGTERM discarded as unnecessary. (3.4.2)
2) Assignment of "SYSTEM" codes is added. (3.4.2)
3) Section 3.4.3 added, including definitions of FINDJCW and PUTJCW intrinsics. (3.4.3)
4) SAVEJCW command and SETJCW command reworked into a different SETJCW command. (3.4.5)
5) Block IF command limited to 15 levels of nesting. (3.4.6)
6) SHOWJCW command added. (3.4.7)
7) CONTINUE command definition altered. (3.4.8)

The Job Control word, the SHOWJCW and SETJCW commands, and the Block IF commands give the user an effective, if minimal, capability to control the flow of execution through his job. The SAVEJCW and SETJCW commands and the JCW provide means whereby status information can be passed and saved, and the Block IF statements allow that information to be examined and acted upon. Thus these few constructs supply the four major elements required for job control.

USER DEFINED COMMANDS - A CL Procedure Facility

Although possibly the most exotic feature proposed for the CLI, the user defined command facility may well become very valuable for all classes of users. Perhaps most important is the fact that it would allow more experienced programmers and designers to create and control the command language environment for other, perhaps non-programmer, users of the system. Secondly it would make it easier and cheaper to extend and tailor the command set of the CLI to fit the specific needs of a user or a class of users. As a consequence it would become easier for OEM's and others to give the system a "customized" feel without too much trouble or HP involvement. On a more mundane level, it would give programmers like us who are involved in program development the ability to execute "canned" sequences of commands with one command, or even automatically at logon, within our sessions (as opposed to stream files, which act outside our sessions). Other uses for this facility undoubtedly will be found as people begin using it.

A user defined command is, in essence, a procedure built from CI commands. This procedure can be created, modified and purged by anyone familiar with the editor. Such a command may be invoked by anyone, even though that person may not know he is using a user defined command. Its effect when invoked is to temporarily switch the source of MPE commands to the command catalog which holds the definition of that command.

5.1 Creation and Management of User Defined Commands

User defined commands reside in ASCII disc files with no special characteristics. There may be several such files for any given group, account and system. There are no restrictions on file names, line length or sequencing information. When more than one user defined command resides in the same catalog, each must be separated from its neighbors by one or more lines with an asterisk in the first byte. The rest of such delimiter lines will be ignored and may be used for commenting purposes. When the user references the file name by establishing the file search hierarchy, the command interpreter will search each catalog and establish a symbol table entry for each unique command. That hierarchy is established with the command

:SETCATALOG [<catalog file name>][,<catalog file name>...]

Embedded system commands are considered last. Among other uses, this allows the system management to limit the use of certain commands by creating a dummy command with the same name in a lower catalog, occluding the name of the restricted command.

Commands may be added, modified and deleted by texting the appropriate command catalog into the editor, editing that file, then reestablishing the catalog hierarchy. It should be noted

that the catalogs will be opened EAR access. This implies that one will not be able to keep one's modified catalog under the same name unless one is careful to see that it is not in use.

The primary advantage of this scheme for managing user defined commands is that it is simple. Since there would be no special rules, file types or unusual subsystems to be learned, training should be minimal. As a side benefit, implementation would be straightforward. However, there are two significant disadvantages. From an implementation point of view, processing will not be as fast as it could be if UDC's were preprocessed into an internal form. The other disadvantage is that it will be difficult to syntax check the command without actually executing the command. On the whole, the simplicity to be gained seems to outweigh the disadvantages.

5.2 Syntax and Semantics of User Defined Commands

User defined commands have two parts, the header section where control information is described and the section containing the command body. The four types of information provided in the header are the command name, the parameters, the execution options, and the natural language description of the command for the HELP mechanism. The command name and the parameters are combined in the first line to form a prototype of the command call. The command body is simply the set of CI commands to be executed when the command is invoked.

5.2.1 The Prototype Call

The first line (including continuation lines) in a user defined command supplies to the command interpreter the name of the command and any parameters along with their defaults. Its form is much as a call to the command would appear, hence the title of this section.

The name of a user defined command may be any alphanumeric string starting with an alphabetic character. Parameters follow the name on the same line (or continuation lines) and are optional. Thus an absolutely minimal first line could be

```
B
```

In this example, the command would be named "B" and would have no parameters.

The name given a parameter in this first line is the formal name by which the parameter is known in the body of the command. Any occurrence in the command body of this name preceeded by an exclamation point (!) will result in the name being replaced by the value of the actual parameter in the command invocation. If a default was supplied in the declaration of the parameter, that parameter is optional for the user of the command. Otherwise it must be supplied when

the command is executed. In this way all parameters have defined values by the time the command body is executed.

The rules for declaring the parameters are more easily explained by example than in English.

```
B PARM1, PARM2=FTN06, PARM3="REC=40,16,F,ASCII"
```

In this example the command name is B. The first parameter to B has no default value and so must be supplied each time B is called. The second parameter, PARM2, has the default value FTN06. The third parameter, PARM3, has the default value REC=40,16,F,ASCII. Since it was bracketed by quote marks, all special characters between the quote marks are included as part of the default value. Since the second parameter had no such quote marks, it has only the five alphanumeric characters FTN06. Stated more generally, the rules for declaring a default value for a parameter are as follows. A default value is optional. The presence of a default value is indicated by an equal sign being the first non-blank character following the parameter name. If a default value has embedded special characters, the value must be bracketed by quote marks. Incidentally, the parameter names may be any string of alphanumeric characters, started by an alpha character. The names used in this example were chosen for purposes of illustration.

When invoking a user defined command the user must follow similar rules. The parameter(s) must be supplied in either keyworded or positional form, if any parameters are required. If the parameters are keyworded, the keyword is the formal name of the parameter (in the definition of the command) and may appear in any order. Positional parameters must appear in the order they were declared and may not be keyworded. If a required parameter is not supplied and the caller is in interactive mode, the user will be prompted using the formal parameter name. If the caller is not interactive, the job will be terminated with an error. Optional parameters for which the caller wishes to use the default are specified by the absence of that parameter in the calling sequence.

Example 1:

```
:B FTN05
```

Inside command B (see the previous example for the command header), the required parameter PARM1 would have the value "FTN05", the optional parameter PARM2 the default value "FTN06", and the optional parameter PARM3 the default value REC=40,16,F,ASCII.

Example 2:

```
:B FTN05,FTN77
```

Inside the command, PARM1 would have the value "FTN05" and PARM2 would have the value "FTN77", since the call is with positional parameters. The default value for PARM2 would be ignored since the caller supplied a new value. The default value for PARM3 would remain the same since no third parameter was supplied.

Example 3:

```
:B PARM2=FTN77, PARM1=FTN05
```

This example illustrates the call using keyworded parameters. Note that this call is identical in effect to the call in example 2.

Example 4:

```
:B FTN05,,"REC=-88"
```

This example illustrates the use of quotes when one wishes to pass a parameter which has embedded blanks or special characters. Note that the second parameter is missing. The default value of FTN06 will be used.

5.2.2 Options

The option part of the command definition affects control of user defined commands. Options allowed are:

```
LIST
LOGON
NOHELP
NOBREAK
```

LIST controls the listing of the procedure body on $STDLIST. The default is for no listing of the body. With OPTION LIST, each time a command is executed, the text of that command, as modified by parameters, will be listed.

LOGON specifies that a UDC will be executed at logon before the CI issues its prompt. Among many other uses, this allows automatic set up of file commands and can be used to put a session into a subsystem automatically.

NOHELP disables the listing of a UDC from the :HELP command. OPTION NOHELP causes ':HELP string' to ignore the possibility that 'string' might be a UDC definition.

NOBREAK disables break for the duration of a UDC. The default is for break to be enabled during the execution of any UDC. If break is hit during a UDC and break is enabled, then if control rests with the CLI, the currently executing CL command will be broken and the user defined command body will be exited permanently. If control rests with a user

program, break will have the usual effect of temporarily returning control to the CLI. The user at this point can do anything normally allowed in break. RESUME resumes the temporarily broken UDC, while a run or ABORT exits the UDC.

5.3.2 HELP in User Defined Commands

If the user requests HELP for a user defined command, the CI will locate and print the appropriate catalog entry:

```
:HELP B
 ------

USER DEFINED COMMAND:

B PARM1, PARM2=FTN05, PARM3="REC=40,16,F,ASCII"
OPTION NOBREAK,LIST
FILE INPUT=!PARM2,OLD
FILE LISTFILE=!PARM1,NEW;!PARM3;DISC=1000,16,2
RUN ABC
:
```

Help in a UDC can be occluded by OPTION NOHELP. In this case the CI HELP system will be called.

5.2.4 Command Body

The command body may include any legal CI command, including other user defined commands. In the latter case, the search of catalogs is strictly up the heirarchy from the catalog in which the current command resides. The body of a command may not contain non-command material, including data for subsystems and programs. In other words, the body of a user defined command is simply an alternate source for MPE commands, not a redirected $STDIN.

5.2.5 Examples

Example 1:

```
B PARM1, PARM2=FTN05, PARM3="REC=40,16,F,ASCII"
OPTION NOBREAK
FILE INPUT=!PARM2,OLD
FILE LISTFILE=!PARM1,NEW;!PARM3;DISC=1000,16,2
RUN ABC
```

This example illustrates the use of the parameters inside the body of the command. If the call had been

```
:B  FTN77
```

the effect would be the same as if the user had entered the commands

```
:FILE INPUT=FTN06,O
:FILE LISTFILE=FTN7
:RUN ABC
```

Note that the reading of
terminated when an end of
asterisk (*) in the first

Example 2:

```
J51MERGE NEWPATCHF,
FILE MERGEE=J514002
FILE MERGER=!OLDPAT
FILE MERGED=!NEWPAT
RUN PMERGER
SPL !NEWPATCHF,J51,
```

Example 2 illustrates one
user defined command faci
a way to do SPL compiles
usual one. Rather than
code, he wrote PMERGER,
the two patch files. The
the user, who may not be
file designators and such
command, it can look like
was developed to satisfy
invoke this "new compiler
just the critical files.
using SPL and CROSSREF.
designer can create comma
in the user's terms. Mos
the 3000 have meaning on
of the 3000. Most of the
such users as data entry
objections to computers
translate their requests
think to the abstraction
thought. Not infrequent
translation effort is re
manual routine required.
implementations staff of
translations of common a
non-programmers to expre
be to them a more natura

Example 3:

```
JS1MERGE NEWPATCHF,OLDPATCHF=MS1,COMPARMS="USLIMIT,CODE"
FILE MERGEE=MS1S002A.HP32002.SUPPORT
FILE MERGER=!OLDPATCHF                          ##old patch file
FILE MERGED=!NEWPATCHF;SAVE                     ##new patch file
RUN PMERGER
IF (JCW=OK) THEN                    ##merge was satisfactory
   SPL !NEWPATCHF,MS1,,MS1S002A.HP32002.SUPPORT;!COMPARMS
ENDIF
```

This is a slightly rewritten version of example 2 which illustrates the use of control information passed back by the program PMERGER. I assume that PMERGER set the JCW based on the results of the program, as indicated.

The User Defined Command facility would allow the user to completely manage his command set. The user could create commands specialized to the needs of a specific group of users without affecting other users. He could occlude system commands he does not want a particular class of users to use. Commands he created could have positional or keyworded parameters, optional parameters and default values, limited side effects, simple rules for invocation, and hierarchies of command sets. These user defined commands would be easily created, edited and deleted. In sum, this facility would provide a powerful, easily used capability to the user through which he could make his system look as if it were tailored to solve his specific problems.

6. MISCELLANEOUS FEATURES

There are several miscellaneous new features, most of which simply clean up some old oversights or are otherwise difficult to classify into one of the preceeding sections. There is no coherent order in which they are taken below.

6.1 Comments

For ease of documentation of batch jobs and user defined commands, comments should be allowed on the same line as commands. Unless there is some good reason not to, comments will be delimited by two hash marks (##). Similarly, we plan to allow sequence numbers.

6.2 Logon User Defined Commands

A means will be added whereby the user may specify a user defined command to be run before he is given control by the CLI. This could be useful for initializing a user's environment automatically. For example, most of us use the same file equates time after time for the line printer, tape drives, etc.

7. UNCL SUMMARY

This specification for an enhanced command language undoubtedly will change somewhat as problems are recognized. In particular, I expect that some of the comments on this report will result in some degree of change. Certainly anything as ambitious as this must evolve. None the less, I believe that each of the major features described is desireable, even necessary, if we are to continue to penetrate the lower cost, less technically oriented market. The user aids would lower the level of knowledge required to use the command language as an effective tool. The flow of control functions would ease the development of those large batch jobs so dear to the hearts of commercial users. Finally, the user defined commands would potentially improve the productivity of programmer and non-programmer alike. A command language with these general features would be a substantial step forward in the development of user oriented systems.

PRIVATE VOLUMES

OVERVIEW

The private volume facility will allow users who have private volume capability to access and otherwise make use of removable disc volumes. The facility has been designed so that the accounting structure, existing as part of MPE, will provide the basis whereby private volume storage space is partitioned and the access to it is controlled. For this reason, a user need not know of or be concerned with the mechanics of volume allocation or mounting; the accessing of a file by the use of a fully qualified designator will cause the system to access, and mount if necessary, the appropriate set of volumes on which the file resides.

Individual removable volumes can be combined to form logical units in the form of volume sets or volume classes. These are the units which will be referenced when access to a private volume is requested. Before any such request can be granted, the volume set or volume class must be first have been defined and made known to the system. This definition can be made at any time by a user with sufficient capability; the system will remember the definition as an aid to future references.

Each group in the accounting structure will be assigned a home volume set. This is the volume set on which the files belonging to the group will be located; it is the set which would be referenced by a user logging on under the group. The set need not be mounted until such time as the user attempts to access a file on the set. At that time, a request for the system to mount the set would be generated. A user can explicitly request that a volume set be mounted before any file access is attempted and can release the set after he is through. User-related mount requests may cause the directory and file space on the volume set belonging to a particular group to be bound to the system's accounting structure. Such binding may occur automatically or by explicit user direction.

Under private volumes, the disc drives configured into the system will be divided into two classes, or domains: system domain devices and non-system domain devices. The distinction between the two classes, simply stated, is that system domain devices cannot be used as private volumes and non-system domain devices can. Specifically, system domain devices are those which do not have removable packs or which the system manager does not desire to have used as private volume devices. Non-system domain devices must have removable packs. Private volumes will initially support the 7905 and 7920 disc drives. It is assumed that successors to the 7920 will be supported also as they are developed.

When a request to mount a volume set has been generated, the system, with the aid of the operator, will determine where the

volume set is to be placed or if the request is to be granted at all. Multiple users may use a volume set concurrently.

DISC DOMAIN DEFINITION

Under private volumes, the disc devices on a system will be allocated to either of two groups. One group, system domain devices, will contain all drives which the system is to consider as permanently mounted, i.e. they are not to be used for the mounting and dismounting of private volumes. This domain will consist of those drives not having supported, removable surfaces and those which may have such surfaces but which are not to be used under private volumes as determined by the system manager.

The second group, the non-system domain (NSD), consists of those devices suitable for use under private volumes. These devices have removable surfaces of the type supported by private volumes and have been allocated by the sytem manager for use as private volume devices. Initially, only 7905 and 7920 disc drives can be placed in the NSD; provisions for successors to the 7920 will be made as the new devices become available.

Domain allocation is made dynamically at each cold-load of the system; the allocation will be made on the basis of the contents volume table rather than on the basis of the particular discs which may be configured into the system. Specifically, each volume added to the volume table will be assumed to reside on a system domain device. Each such volume must be present when attempting to cold-load the system but it may reside on any suitable drive. An exception exists as to the volume first assigned to ldev 1 - that volume must always reside on ldev 1, it cannot be moved. After it has been determined that all volumes indicated by the volume table are present on the system, all other discs which have been configured into the system and which, at the time of the cold-load, are either offline, or online but do not have mounted on them a system volume, will be considered as non-system domain drives and will be available for private volume use.

During a cold-load operation, the following console messages will be printed for each physically mounted, non-system domain device:

```
NON-SYSTEM VOLUME ON DEVICE <ldn>
ADD TO SYSTEM VOLUME SET?
```

If YES is entered in response to the second message, the volume will be added to the system volume set. The operator will be prompted for the volume name and pack size of the volume. The volume's new name should then be added to the volume table.

3. VOLUME SET/CLASS DISCUSSION

3.1 Volume Set/Class Definition

3.1.1 Volume Set

A volume set is a set of volumes which share a common file directory. A volume set is limited in size to eight volumes; its member volumes are not restricted to being of the same type, i.e. a volume set may consist of a mixture of volumes mountable on any of the supported drive types. Members of a volume set are unique to that volume set and cannot be shared among other sets. Those volumes, permanently mounted on system domain devices will be known as the system volume set. The system volume set will be named, by the system, with the volume set name, SYSVS.

Volume sets and classes will be assigned and allocated at the group level of the accounting structure. Each group will be assigned to one and only one logical volume set - its home volume set. All files belonging to the group will reside on, and only on, the home volume set. However, a particular group may have its files residing on more than one physical volume set as along as the set has the same name and otherwise satisfies the home volume set definition. This feature is used when dealing with different generations of a home volume set. More than one group or account may share a volume set; a volume set may be the home volume set to more than one group in one or more accounts. A group can have only one home volume set at any one time (i.e., a group cannot span different volume sets).

Every volume set will have its own accounting directory and will reside on the master volume (Sec 4.3.1...1 Master Volume) of the volume set. Each and every directory will be indentical in their data structure, but not necessarily in their content. This uniformity of data structure allows all such directories to be searched and maintained by one set of directory routines. The system volume set's directory will hereafter be referred to as the system directory. Similarly, a non-system volume set's directory will be referred to as a non-system directory. Only account and group entries will be allowed to coexist, when appropriate, in the system directory and non-system directory simultaniously. Other directory entry types, user and other entry types to be disussed later, will not be maintained in a non-system directory since only account and group entries are necessary in searching for and maintaining file entries. File entries will exist only on the home volume set to which the file's parent group has been assigned to at that time. The system and non-system directories become bound to one another at the file level when a logical mount request for a home volume set on behalf of a group is invoked.

1.1.1 Master Volume

The master volume is that volume of a volume set, common to all volume classes defined for the set, which contains the volume set's accounting directory. It also contains a volume table of volumes defined as members of the volume set. The master volume of the system volume set is defined to be mounted on level 1.

1.2 Volume Class

A volume class is a proper subset of the volumes in a volume set. It must include the master volume of the set. It is accessed as a unit and identified by a unique volume class name. Most importantly, it is the smallest user-referenceable and logically mountable volume unit under this facility. A single volume can be specifically referenced or logically mounted only if it is itself a volume class (and necessarily also the master volume of a volume set). A volume class is therefore a shorthand method of notifying the system that only a portion of a defined volume set need be mounted to satisfy a user's anticipated set of file-access requests. The essential feature/requirement of a volume class is that it must fit within the drive set allocated to the NSD. Hence, all volumes of a volume class must be concurrently mounted and concurrently mountable.

A volume set will always be treated as a volume class until such time as it is expanded beyond the capacity of the NSD. Thereafter, each file operation must specify a volume class which represents a subset of the members of the set. A user must begin defining and using volume classes once the NSD has been exceeded by the volume set definition. Alternately, a user may avoid the use of volume classes as long the volume sets he uses do not exceed the capacity of the NSD. A user will be warned whenever his volume set definition (via the :DEFVSET and :ALTVSET commands described below) results in a specification which exceeds the number or types of volumes in the NSD.

The following is a summary of volume class characteristics:

1. A volume set will be treated as a volume class for all purposes until such time as its definion exceeds the capacity of the non-system domain.

2. All volume classes in a volume set must have as a member the master volume of the volume set.

3. The master volume is the only volume which can be shared by different volume classes of the same volume set unless one volume class is a proper subset of another.

4. All the volumes of a volume class must be concurrently mountable.

5. All the volumes of a volume class must be concurrently mounted.

6. A file cannot exist on more than one volume class unless the members of one volume class are a proper subset of the members or another.

3.2 Volume Set/Class References

A volume set or volume class reference, when fully qualified, is in the form

```
    *
{         }.<group>.<account>
 <vcsid>
```


where vcsid refers to a previously defined volume set or class and "*" implies the home volume set for the group and account specified. This format provides the identifiers necessary to adequately search for and find a specific volume set definition entry in the system directory. When <vcsid> is specified, <group> and <account> refer to that of the creating group and account of the volume set. No ownership or exclusive access is implied; group and account merely aid in the searching of the directory for the appropriate volume set definition. These entries are described in the section, Volume Set Definition Entities (4.3.3). A volume set or volume class can be accessable to multiple accounts and thus multiple groups (i.e. different accounts can share a volume set).

A non-system volume set may be used on a host system only after the accounts, groups, and users for the files to which access is desired have been installed and the appropriate account manager (Sec. 4.3.4.2) has created the volume set definition through the :NEWVSET command as discussed below. Thus, when transporting a volume set to another system, the group, account, and volume set definition must be created on that system before the set can be successfully mounted and used.

Some directory maintenance operation will require that at least the volume set's master volume be pre-mounted before the operation can be successfully performed. The following is a summary of those command with their specified parameters.

1. :NEWACCT and :ALTACCT with VS parameter specification.
2. :NEWGROUP and :ALTGROUP with VS parameter specification.
3. :PURGEACCT and PURGEGROUP with VS parameter specification.

A more complete description of the purpose and operation of these commands will be outlined in their appropriate sections.

.3.3 Volume Set Definition Entities

These entites will contain a volume set's definition. The attributes of a volume set definition is its name, the name of its members, the disc sub-type requirements of each member, and the definition(s) necessary for the regrouping of its members into subsets (volume classes).

These definitions are to be new entities in the existing directory structure and will be descendants of the group and account in which they have been defined.

.3.3.1 Multi-account volume set

Any volume set can be the home of multiple account, i.e. there is no restriction (except for directory space considerations) to the number of accounts a volume set can house.

.3.3.2 Home Volume Set

A home volume set is that volume set assigned to a group when the group is either created or altered via the :NEWGROUP or the :ALTGROUP commands. It is not necessary that a group's home volume set definition be defined in that same group and account. The definition can be created in any account and in any group of that account so long as the creating user has CV capability (sec. 1.3.4.2). The default volume set for groups and accounts not using private volumes will be SYSVS (described later).

.3.4 Volume Set Management

All the volumes of a volume set need not be mounted in order to access the volume set. This allows the number of volumes in a volume set to exceed the number of physical drives available for mounting of removable volumes as defined by the set of MSU devices. However, as all the volumes of a volume class must be concurrently mounted, the maximum file size on a non-system volume set is limited to the capacity of the physical drives in the MSU.

3.4.1 Volume Set/Class and Volume Name Relationships

Volume names within a given volume set need not be unique with respect to either the parent volume set name, to volume class names within the parent volume set, or to the volume/volume class/volume set names outside of the volume set. However, the volume class names within a given volume set must be unique with respect to other volume sets and classes defined by the group and account. Volume class name uniqueness is not required as to volume/volume class/volume set names defined by other groups and accounts.

There is a significant distinction between a volume name and a physical volume. Many physical volumes in a given installation may have the same volume name, each belonging to a different volume set.

All console messages, and operator commands and replies which relate to volume mounting will contain a volume set or a volume class name. Every removable volume should have an external label denoting both volume set or volume class name(s) and volume name.

3.4.2 Volume Set/Class Creation

Only system managers, and account managers with CV capability (sec. 4.3.5.3) will be allowed to create, alter, or purge volume set definitions.

Volume sets and volume classes are defined by appropriate account managers through the :NEWVSET user command. The only exception is the system volume set, SYSVS which is defined at cold load time and includes all the drives in the SD. Once a volume set or volume class has been defined, the definition is kept on disc in a volume set definition entry. These entries enables the system to remember which volume sets have been defined. Thus, a system or account manager need not define a particular volume set or class more than once. The volume set definition will be created in the log-on group and home account at the time of its creation. A volume set definition will remain in the directory until explicitly deleted by a either a :PURGEVSET, :PURGEACCT, or :PURGEGROUP command. A physical volume set is not created at the time of its definition. Rather, only its members and thier corresponding storage types are defined to the system. A formatted list of defined volume sets will be available to the user through the :LISTV user command.

The syntax of the :NEWVSET command is as follows:

```
:NEWVSET vsname
 ;MEMBERS= vname:type [,vname:type ... [vname:type]]
 [;CLASS= vcname:vname [,vname] ... [,vname]]
```

one vname of the CLASS and/or MEMBERS list must have the name vsname. This volume will be designated, by the system, to be the master volume of the specified volume class and/or volume set.

The system will implicitly reference vsname and vcname respectively as vsname.groupname.accountname and vcname.groupname.accountname, where groupname and accountname are the logon group and account.

The :NEWVSET command allows the following items to be specified:

1. The volume set name (up to 8 alphanumeric characters beginning with an alphabetic). The system volume set will be predefined by the system, to the name: SYSVS.PUB.SYS. Any number of non-system volume sets may be defined. Volume set names need not be unique between accounts and groups. The name of the volume set will also be given to the master volume of the set. An appropriate user can thus create a single-volume volume set by simply entering

   :NEWVSET vsname; MEMBERS= vsname:type

   The name of the volume set will be magnetically recorded on the master volume.

2. A maximum of eight volume names may be specified in the list. These are the volumes comprising the volume set. Specified with each volume name is a volume type which defines the type of drive required to accommodate the volume. Within any volume set, all volume names must be unique. A volume name is recorded (magnetically) on each volume in every volume set. Any given physical volume has exactly one name.

3. Optionally, any number of volume classes may be defined local to the volume set. Each volume class consists of a volume class name and an associated list of one or more volume names. The list of volume names must be a proper subset of the volumes comprising the volume set itself. In addition, a volume class definition must always include the volume set's master volume, and the number and types of volumes in the class must not exceed the system's ability to satisfy a mount request from the pool of MSD devices. All volume class names must be unique with respect to the set of volume set and volume class names defined by the user or account manager.

3.4.3 Volume Set/Class Modification

Volume set definitions may be modified by the :ALTVSET command. This command can be used to increase the size, i.e., the number of member volumes, of a previously defined volume set; however, a volume set, once defined, cannot be reduced in size. The format of the command is

```
:ALTVSET vsname
 [;EXPANDCLASS= vcname:vname [,vname] ... [,vname]]
 [;ADDCLASS= vcname:vname [,vname] ... [vname]]
 [;EXPANDSET= vname:type [,vname:type] ... [,vname:type]]
```

3.4.4 Volume Set/Class Deletion

Previously defined volume sets may may be deleted through the :PURGEVSET command. This command may be entered regardless of whether any member of the set is physically mounted as long as no member is being used (i.e. logically mounted). The syntax of the command is

```
:PURGEVSET vsname.
```

A volume set's definition can also be purged, implicitly, by issuing the :PURGEGROUP and :PURGEACCT commands.

A volume set definition and/or its parent group and account will not be permitted to be purged as long as the definition remains in an un-use state; i.e., the associated volume set or volume class is mounted (physically) and is in use (logically mounted).

3.4.5 Volume Set/Class Transportability

Transportability between systems will require that a transported volume set be first defined by the appropriate system or account manager via the :NEWVSET command.

3.5 Installing Accounts and Groups

The concept of installing and deleting accounts and groups in either the system directory or in the non-system directory is operationally partitioned as two distinct functions. The following discussion describes the directory maintenance operations with regard to non-system directories.

As stated in section 4.3.1.1, Volume Set, each volume set will have its own accounting directory. A directory data structure, with a null entry state, is created when a volume set's master volume set is initialized by the VINIT subsystem. The means for the creation of accounts and groups in these directories will be facilitated through extensions of the NEWACCT, ALTACCT, NEWGROUP, and ALTGROUP commands. An additional parameter (VS=volset) and subparameter (SPAN) have been added

to allow the duplication of entries (bridging entries between system and non-system directories) in the non-system directory. The VS parameter allows a volume set designation to be specified. This specification will then be referenced for determining the location of and maintaining control over the specified volume set. The SPAN subparameter indicates that the entry, account or group, should be created in the non-system directory. Similarly, the PURGEACCT and PURGEGROUP command have been extended to include the VS parameter for the purging of account and groups from non-system directories. The directories are mutually exclusive of one another with regard to the purging functions, i.e. purging accounts and groups from one directory has no effect on the corresponding entry (if it exists), in the opposite directory. The prerequisite for the invokation of the NEWxxx and ALTxxx using the SPAN subparameter and the PURGExxx commands is that the appropriate volume set be premounted via the MOUNT command.

Creating account and groups in the system directory are as currently defined in the System Manager/Supervisor Manual.

Section 4.6.1.2.4 contains the syntax for the extended NEWxxx, ALTxxx, and PURGExxx commands.

4.5.1 Accounts

The AM, if possessing CV capability, will be allowed to define volume sets, via the :NEWVSET command. But as stated previously, these definitions are system wide accessable for assignment by any user possessing CV capability.

The account manager or any user with CV capability can assign volume sets to groups within the account so long as a group's allocated file space is null when the group's old home volume set is the system volume set.

The assignment of either a specific or the implied (by omission of specific assignment) SYSVS home volume set to a group will be in name only until at least one file allocation operation is performed on behalf of the group. It at this time that the group is bound to its home volume set (system or non-system.

4.5.2 Groups

The AM issues the :NEWGROUP command.

:NEWGROUP groupname [;VS=volset [:SPAN]]

The command creates GROUP=groupname under the account of the AM.

If the VS parameter is omitted, the group groupname is temporarily assigned to the system volume set, SYSVS, otherwise, it is temporarily assigned to the volume set, volset. The concept of of when a group becomes bound to its home volume is described in section 4.3.5.1.

Similarly, the AM can issue the

:ALTGROUP groupname [;VS=volset [:SPAN]]

to assign a volume set, and under the same rules, as described above.

3.5.3 Users

Use of the Private Volumes facility will be restricted to those users who have sufficient capability. Two new capabilities will be available under this facility:

1. Volume set creation (CV) capability and
2. Volume set usage (UV) capability

The CV capability will allow the account manager to dynamically create new volume sets or to extend currently defined volume sets to include additional members. The CV capability implies the presence of the UV capability; both need not be specified when CV capability is given. Volume sets are defined by the use of the :NEWVSET command and modified by the :ALTVSET command as described above. It will be recalled that the system remembers volume set definitions given by the :NEWVSET command in a volume set definition entry thereby eliminating volume set re-definition at each logon.

The UV capability allows a user to make use of currently existing volume sets but does not, in itself, allow the user to create new volume sets or to alter the number of members of a currently existing set.

Any volume set created or defined by the account manager may be used and shared by those users in the account who are given UV capability.

NOTE: With the above definitions, notice that old accounting card decks can be used without change to create the accounting structure in which all groups are assigned to the system volume set.

3.6 Disc space allocation under PV

6.1 User file allocation

Today's method of allocation is by the device parameter of the FOPEN intrinsic which may be overridden by the DEV parameter of the FILE and BUILD commands. This parameter may specify a logical device number (ldn) or a device class name. A device class (dc) is a collection of one or more ldn's. If the device parameter is omitted, its default value is the device class name "DISC". It is the responsibility of the system manager to define all device classes in the system. There is no MPE-defined device class in any system; MPE only assumes that device class names DISC and SPOOL must exist under certain conditions.

The discussion that follows applies not only to the FOPEN intrinsic (and its associated :FILE command) but also to the :BUILD command, which has an identical DEV parameter. The :BUILD command simply issues an FOPEN with file equation disallowed.

Under private volumes, the device parameter will have the following form:

```
          ldn
          dc
DEV =     #
          #vsname
          ##volname
```

where ldn and dc retain their old meanings but with new qualifications.

The following internal algorithm is used for the allocation of all new files:

1) If the ldn or dc specifies a non-shared (i.e., non-disc) device, then allocate the device in the usual way so that the remainder of this sub-section does not apply.

2) Form a fully-qualified actual-designator for the file. Determine the volume set assigned to the FOPEN group (and account). This will be the the home volume set (hvs).

3) If the device parameter specified a ldn, the file will be allocated to the ldn only if one of the volumes in the home volume set currently occupies the ldn (i.e., the intersection of the device parameter ldn and the home volume set's ldn's)

4) If the device parameter specifies a device class, the file will be allocated within the home volume set's volumes which currently reside on ldn's bounded by the specified device class (i.e., the intersection of the device class ldn's and the home volume sets's ldn's.)

5) If DEV=*, then the file is allocated to any of the volumes of the home volume set. (If the DEV parameter is omitted, the default is DEV=*.) If DEV=*vcname, and that vcname is a member of the home volume set, allows the file to be allocated to any of the volumes within the volume class. If DEV=**volname, and that volname is a member of the home volume set, allows the file to be allocated to that volume.

NOTES:

1) Any given file may span the volumes of a volume set, but all extents of a file are confined to at most one volume set. In other words, regardless of how a disc file is opened its extents will never be permitted to cross volume set boundaries.

If a file is opened by DEV=*, then the extents are confined to the volumes of the volume set. If the file is opened by DEV=ldn or DEV=**volname, then all extents of the file are confined to the single ldn or volume. If the file is opened by DEV=#c, then the extents of the file are confined to the volumes of the home volume set which reside on the ldn's bound by the device class. If the file is opened by DEV=*vcname then the extents of the file are confined to volumes within the volume class.

2. So that old programs will continue to run, it is recommended that system managers continue to assign all disc ldn's to device class=DISC, with the possible exception of ldn=1.

3) It is recommended that users avoid the ldn and device class forms of the FOPEN device parameter (for disc) unless they really know how devices, and device classes are configured for a particular installation.

3.6.2 Allocation of space for the device class SPOOL

Only logical devices in the system (disc) domain may be assigned to device class SPOOL.

There is no reason to allocate spoolfiles on the volume set assigned to the creating user because (1) users cannot save spoolfiles because spoolfiles have no directory entries in the current version of MPE, and (2) doing so would prolong the ASSIGNED period for those ldn's occupied by the volume set.

VOLUME SET MOUNTING

Volume set mounting refers to the logical attachment of a particular volume set to the set of disc devices recognized by the system. This differs from volume mounting which refers to the physical placing of a volume on a spindle.

A volume set will not be considered mounted by the system merely because one or all of its member volumes is physically mounted. Rather, a volume set mount request must first have been received by the system and ultimately granted by the operator.

Whether a volume set mount request is satisfied or rejected depends to a large extent on the individual states of the discs in the non-system domain. At any point in time, a disc device will be in one of the following three states:

1. DOWN - The logical device is out of service. A non-system disc in the DOWN state is not a candidate for assignment to a volume set.

2. AVAILABLE - The non-system device is not in use and it is available for assignment to a volume sets. A system disc can never be in the AVAILABLE state. The fact that a volume may be physically mounted on the drive has no effect on the drive's AVAILABLE status.

3. ASSIGNED - A volume of a volume set is currently assigned to this logical device, and the device is not a candidate for re-assignment.

The means from progressing from one state to another differ according to whether the disc is a member of the system or non-system domain. When the system is first brought up, system discs are placed in the ASSIGNED state and non-system discs are placed in the AVAILABLE state. The latter is true irrespective of whether a volume is physically mounted on the device.

System domain devices can change state only by means of the operator =UP and =DOWN commands. Non-system domain devices can change state by either the use of these same commands or by volume set mount/dismount operations.

.1 UP/DOWN Console Commands

The console commands =UP and =DOWN apply to both system and non-system discs. However, because system discs can be in only two of the three defined states, the =UP and =DOWN commands have slightly different meanings for system versus non-system discs.

A DOWN command entered for the master volume of a mounted volume set will cause further requests to use the set to be rejected and, when the last user has dismounted it, to prevent any user from using the set. If a user domain drive not containing a mounted volume set member is DOWN-ed, it will not be considered as a candidate for Private Volume's use until it is UP-ed.

..2 Mount Initiation

All mounting/dismounting of volume sets is confined to discs in the non-system domain.

In the general sense, there are two ways in which mounting can be effected:

1. Operator-initiated mounts via the =MOUNT console command. The devices referenced by a given operator-initiated mount request will remain in the ASSIGNED state until the operator issues a corresponding =DISMOUNT command.

2. User-initiated mounts arising under the following circumstances:

   a. A user :MOUNT command within a job or session.

   b. A user command, e.g. :BUILD, requiring the presence of the home volume set.

   c. Programmatic access to a file whose volume set is not mounted.

An important feature of user-initiated mounts is the binding of directories which may occur. Such binding occurs automatically for implicit user-initiated mounts and upon request for explicit user-initiated mounts. In the latter case, the user can request that binding is to occur at the time the :MOUNT request is satisfied. Whether implicitly or explicitly generated, a user-initiated mount will cause the directory information on the volume set to be linked into the system's accounting structure thereby binding the volume set to the system. Such binding will only occur with respect to the group for which the mount request was generated. That is, in a multi-group volume set, binding will occur only as to those groups which have been explicitly referenced in the user-initiated mount, e.g. logon group or FOPEN group.

If a volume set is mounted because of a user-initiated request, the corresponding dismount is effected when the last requesting user has relinquished his hold on the set. In other words, the volume set is not made AVAILABLE automatically when the user who caused the set to become ASSIGNED relinquishes his need for the volume set; there may

be subsequent requestors of the same volume set prior to the relinquishment of the VS by the initial requestor. In a similar manner, the operator can cause the devices associated with a volume set to remain ASSIGNED across jobs by means of the =MOUNT command.

2.1 Operator Control Over Mount Requests

The console operator will be provided with levels of control over the acceptance,rejection, and granting of user-initiated mount requests. Acceptance and rejection of mount requests refer to the filtering of such requests by the system before there is any attempt to satisfy the request; granting will occur when the request has been accepted and a sufficient number of drives of the correct type are available to satisfy the request. The levels of control will be provided by the following console command:

```
          ON [,AUTO]
=VMOUNT   {          } [;ALL]
          OFF
```

ON sets the mode whereby all user-initiated mount requests will be accepted; OFF indicates that all user-initiated requests are to be rejected. A rejected mount request will cause an I/O error to occur if the request was generated programmatically, or a Command Interpreter error to occur when the request was generated directly by the job or session, and, in either case, the following console message to be printed if the ALL option is set:

```
ST/<pin #>/ MOUNT FOR user.account REJECTED
```

The optional parameter AUTO, when specified, will allow the system to select available drives and satisfy the request without operator approval. If not set, the operator will be required to reply to all mount requests, even those requesting the mount of a volume set already mounted and in use by another. The ALL parameter will cause all mount-related console messages, including those not requiring operator action, to be printed on the console. The default modes (from INITIAL) will be OFF and ALL. Thus, all mount requests will be rejected but the operator will be notified on the console of each rejection. VPE will attempt to satisfy the request before asking for operator assistance in selecting a set of drives. If the volume set is physically mounted and the AUTO option is set then the request will be satisfied automatically without operator intervention; if ALL is set, the following in-use console message will appear:

```
ST/<pin#>/vsid.group.account IN USE BY user.account
```

If the volume set if not physically present, the system will begin looking for available drives of the types required to

satisfy the request. If sufficient drives are available, i.e. not in the DOWN state the system will reserve the drives for the request and notify the operator to place the set on the selected drives by the following message:

```
?I/O/<pin#>/ MOUNT vcsid.group.account ON ldnlist
```

The operator must respond with

```
          YES
=REPLY {      }
          NO
```

A NO reply rejects the user's request. A YES reply indicates to MPE that the volumes of vsname have been mounted on ldnlist. If the request is the first for a newly created volume set, scratch volumes should be mounted and then attached to the volume set by being initialized through the :VINIT subsystem (Section 6.0).

If sufficient drives are not available to satisfy the request, the system will reserve those drives which will satisfy part of the request and notify the operator of the deficiency by the following message:

```
?I/O/<pin#>/ DRIVES FOR vcsid.group.account: type(numb)
[,type(numb)]
```

The operator should perform whatever operations required to free a drive of the type required and respond YES when the drives are AVAILABLE; a NO response will cause the previously reserved drives to be made AVAILABLE and the request to be rejected.

If AUTO is not set, the operator will be required to reply to

```
?I/O/<pin#>/ ACCESS TO vcsid.group.account BY user.account
```

This message will be printed whenever AUTO is not set and a user wishes to mount a volume set to which she is not currently attached. The reply opportunity gives the operator complete and final control as to who will have access to Private Volumes.

When the last requesting user, including the operator by means of the =DISMOUNT command, relinquishes a volume set AND THE ALL PARAMETER WAS SPECIFIED IN THE LAST =VMOUNT COMMAND, THE FOLLOWING message will be printed on the console:

```
SI/time/ AVAILABLE DRIVES ON ldnlist
```

At this time, the devices in ldnlist revert to the AVAILABLE state automatically; no operator intervention is required.

After a VS has been mounted by means a user-initiated mount, the operator can issue a =MOUNT console command which will cause the VS to remain mounted even after the last requesting user has relinquished the VS.

If the operator wishes to prohibit further mounting of a currently mounted volume set, he should =DOWN the master volume of the set. The =DSTAT command can be used to determine which volume is the master volume of a physically mounted set. The master volume will have the same name as the volume set itself.

4.2.2 Operator-Initiated Mount Requests

Unlike the user-initiated mount requests discussed below, the operator cannot specify that home volume set be mounted for a particular group and account; no binding of directories will occur on an operator-initiated mount. In addition, unlike user-initiated mounts in which MPE selects the set of discs, the operator may select the discs in operator-initiated mounts.

```
                *
=MOUNT {        } .group.account [;GEN = GENERATION]
       vcsid
```

The volume set to be mounted can be referenced directly by using the actual vcsid of the set or indirectly as the home volume set of the group and account specified by using the "*" construct. Volume set/class references constructs are described in section 1.3.2. Unlike the :MOUNT command discribed below, specification of the home volume set construct will not cause binding of directories to occur for the group specified; the =MOUNT command will never cause binding to occur.

THE MOUNT REQUEST WILL BE PROCESSED IN THE SAME MANNER AS A USER-INITIATED REQUEST; ALL DISCS CONTAINING MEMBERS OF THE WILL BE PLACED IN THE ASSIGNED STATE. THE MOUNT REMAINS IN EFFECT until an =DISMOUNT command for THE SET IS ENTERED.

```
                *
=DISMOUNT {     } .group.account
          vcsid
```

This command directs the system to dismount the volume set as a unit from the drives on which its members reside. THE OPERATOR MUST HAVE PREVIOUSLY CAUSED THE SET TO BE MOUNTED.

4.2.3 User-Initiated Mount Requests

A user-initiated mount request may result in the binding of directories with respect to a particular group. Binding is the process whereby the directory and file space on a volume set belonging to a particular group is attached to the system's accounting structure; after binding, all references (e.g. LISTF) to the files of the group will be directed to those files residing on the volume set. There are three ways in which user-initiated mount requests are generated:

1. Explicit job/session request.

As described in the sections immediately following, a single job step may cause a volume set to be implicitly mounted then dismounted. A user has the option of causing a volume set to be mounted for periods exceeding a single job step by use of the :MOUNT command. A set so mounted can be dismounted by the user by use of the :DISMOUNT command. The use of these explicit control operations would be advisable when the user expects to be accessing or creating files in the same group and account in more than one job step. An implicit :DISMOUNT will be performed for the user for each volume set she has mounted at logoff time.

The :MOUNT command allows a user to request that a volume set be mounted. In addition, she can specify that the volume set is to be treated as a home volume set have it bound to the system's accounting structure with respect to a particular group. The format of the command is

```
                *
:MOUNT [(              )[group [.account]]] [;GEN=genindex]
        vcsid
```

The volume set to be mounted can be referenced directly by using the actual vcsid of the set or indirectly as the home volume set of the group and account specified by using the "*" construct. Volume set/class references constructs are described in section 4.3.2. If the home volue set form is entered, binding of directories will occur for the group specified; if the volume set is directly referenced, it will be mounted but no binding of directories will occur.

The command has two optional paramaters: a group specifier and generation specifier. The group specifier indicates the group whose home volume set is to be mounted and for whom the binding of of directories and file domains is to occur. If it is ommitted, the logon group and account are assumed, i.e. the group specifier need not be entered when the user wishes to access only the files of the group under which she logged on.

The optional GEN keyword parameter indicates to the system which generation of the home volume set is to be mounted. Generation indices are discussed under the COPY function of the VIEW subsystem. Only one generation of a volume set can be mounted on the system at any one time; a mount request which would cause a different generation to be mounted will be rejected.

The :MOUNT command is rejected if there is a pending operator-initiated =DISMOUNT for the spcified vs. The requesting job or session is suspended until the mount is completed or rejected.

A job or session is not allowed to explicitly mount a particular volume set (including a class of that set) more than once at any one time; the request will be rejected with a duplicate-mount error.

The :DISMOUNT command allows a user to dismount a home volume set which he previously explicitly mounted by means of a :MOUNT command. The format of the command is

```
                  *
:DISMOUNT [{          }[group [.account]]
           vcsid
```

If the optional group specifier is not entered, the logon group and account is assumed. A user must use the same form of volume set reference in both the :MOUNT and :DISMOUNT commands for the same volume set; the forms can be intermixed as to different volume sets.

The :DISMOUNT command indicates to MPE that the specified volume set is no longer needed by the requesting job. If there are no other users of the volume set, the devices on which it resides will be returned to the AVAILABLE state. A user can only dismount a volume set which he had caused to be mounted; a user cannot effect the status of the set as to other users.

2. Implicit job/session request.

Various user commands which cause access to the logon group's home volume set will cause mounts to be requested when the commands are entered and the volume set is not mounted. The :BUILD command is an example.

3. Implicit programmatic request.

A user may issue an FOPEN call referencing a file residing on an unmounted volume set: this will cause an implicit user-initiated mount request. The request will fail if there is a pending =DISMOUNT for the specified volume set.

An FOPEN mount is effective until the corresponding FCLOSE is issued. This method would be used when a single job step requires a certain volume set.

In each of the above cases, the user-initiated request is first subject to the mode established by the =VMOUNT command and to the operator's willingness to grant the request.

AUTO-RECOGNITION OF VOLUMES

When a non-system domain drive is in the DOWN or AVAILABLE state, volumes may be physically mounted or dismounted from the drive without adversely affecting the system. An available device taken offline will cause the device to enter the DOWN state and conversely when the device is brought online.

The system will automatically sense when a drive has gone on-line (mount) or off-line (dismount), update the volume table, and print one of the following console messages:

a. SI/<time>/ vname OF vcsid.group.account ON LDEV# ldn

This message will appear when a non-scratch, conditioned volume is mounted.

b. SI/<time>/ SCRATCH VOLUME ON LDEV# ldn

This message will appear after a scratch, conditioned volume has been mounted.

c. SI/<time>/ SYSTEM VOLUME ON LDEV# ldn

This message will notify that a volume which has been previously initialized as a system domain volume has been mounted on a private volume device.

d. SI/<time>/ DISMOUNT OF LDEV# ldn

This message will be printed whenever a non-system domain drive which is not in use goes off-line.

e. SI/<time>/ IN-USE VOLUME DISMOUNTED ON LDEV# ldn

This message will be printed whenever a member of a logically mounted volume set is taken off-line.

f. SI/<time>/ UNFORMATTED VOLUME ON LDEV# ldn

This message will be printed when an unconditioned volume has been mounted.

g. SI/<time>/ UNREADABLE LABEL ON LDEV# ldn

This message will be printed when the volume contains a media error which prevents its label from being read by the system.

DISC VOLUME CONDITIONING

Elements of Conditioning

A disc volume, to be accessable by MPE, must first have gone through a conditioning process. This conditioning process contains three phases. These phases, in the order in which they must be performed, are

1. formatting - the placing of addressing and, possibly, timing information on the disc.

2. initialization - the configuring of Free Space and Defective Tracks tables.

3. labeling - giving the volume a name.

Formatting is normally performed by stand-alone programs (e.g., SUFUTIL) without the help and even the presence of MPE. Initialization is currently performed only by INITIAL when either a RELOAD is being performed or recovery of lost disc space is being attempted. Labeling is performed whenever the presence of an unlabeled volume is detected by INITIAL.

Because of the possible need under private volumes for frequent mounting of new, unconditioned volumes or re-labeling of old, conditioned volumes, the ability to totally condition disc volumes while the system is up and running will be provided. The volumes of the system domain cannot be conditioned under private volumes.

To condition a volume, the logical device on which the volume is to be placed must be a non-system disc in the DOWN state.

6.2 VINIT Subsystem

Volume conditioning will be performed by a user-accessible subsystem. The subsystem will carry on a dialog with the user allowing him specify the sequence of conditioning events. The use of the subsystem will be restricted to those users with operator capability.

The subsystem will be invoked by the :VINIT command. The format of the command is

:VINIT [listdevice]

Specification of a list device will cause output generated by certain of the functions in the subsystem to be printed on the device specified. Once invoked, the subsystem will prompt the user to enter one of a set of allowed commands. The set of such commands will include:

> INIT vname, ldn [,vsname.group.account] [;GEN=genindex]

The INIT function is used to complete the conditioning of a previously formatted volume. If the optional volume set name is not specified then it must have been entered as part of a previous INIT command; an error will occur otherwise.

A volume can be initialized to any desired generation by specifying the GEN optional parameter; if not specified, the volume will be given a generation index of zero. All volumes of a single volume set should be initialized with the same generation index to allow the set to be successfully mounted.

If the volume specified is the master volume of the current volume set, additional initializing will occur. Specifically, a volume table of member volumes and a volume set file directory nucleus will be placed on the volume.

Before a volume can be initialized the following conditions must exist:

1. a volume must be physically mounted on the device function)

2. the logical device must be in the down state (=DOWN or DOWN

3. the volume must be a scratch volume (SCRATCH function).

As part of the initializing process, INIT will construct a Defective Tracks Table on the device. If defective tracks are encountered during initialization, a message indicating the number of suspect tracks detected will be printed. The user should invoke the DTRACK function after completion of the INIT function to dispose of the defective tracks.

> FORMAT ldn

The FORMAT function will allow on-line formatting of removable volumes. Formatting a volume would be necessary only when a new, previously unused volume is to be initialized or when an irrecoverable pack error has been detected on a previously used volume.

> SCRATCH ldn [;RESET]

The SCRATCH function will allow the operator to make a mounted volume scratch and available for assignment to a volume set. The RESET parameter will allow the operator to make a volume previously (and perhaps inadvertently) made scratch non-scratch.

COPY fromldn,toldn [;GEN [=geninndex]]

The COPY function will cause the contents of the from-volume to be copied to the to-volume. The function operates on a volume-by-volume basis. The master volume of the set need not be mounted but it is advisable to copy all members of a volume at the same time.

If GEN is not specified then the generation index will be one greater than that of the from-volume. If specified but an index is not given, the generation index will be the same as that of the from volume. The to-volume will be given the user-specified index it provided in the command. The values for a user specified index can range from 0 to 65535. No error will occur if the user specifies an index less than that of from-volume.

In order for the copy operation to succeed the following conditions must exits:

a. THE TO-DEVICE MUST BE IN THE DOWN STATE OR HAVE A SCRATCH VOLUME residing on it.

b. both devices must be of the same type. Type here refers to the type of drive involved, e.g. 7920.

c. the to-device must have no deleted tracks.

d. the to-device must be a scratch pack.

COPY will place an exact image of the from-volume on the to-volume. No compaction or other re-organization of the file space on the volume will be performed.

COPY will operate on only one from-volume at a time. Thus in order to back-up a multi-volume volume set, multiple COPY operations would have to be performed. As the volume set directory is located only on the master volume of the set, it would be of little practical use to back-up only one member of a multi-volume volume set.

```
            SIZE=n
> COND lun [;       ]
            ALL
```

The COND function causes disc compaction operations to be performed on the volume located on logical device lun. This fuction is not part of the Private Volumes facility and is discussed separately in section 9.

>DSTAT

THE DSTAT FUNCTION HAS THE SAME SYNTAX AND PRODUCES THE SAME OUTPUT AS THE :DSTAT AND =DSTAT COMMANDS.

> PDEFN [* .group.account]
> vsname

The PDEFN function will list the elements contained in the volume set definition directory for the volume set specified. If "*" was entered in place of vsname, the volume set definition of the home volume set for the group specified will be referenced. If neither was specified, the list will refer to the volume set specified by the last INIT function. The output provided by this function will be useful to the operator in directing the sequence of INIT operations necessary for multi-volume volume sets. The output will list the following information:

1. volume names with the master volume listed first

2. drive type, e.g. 7920.

> PLABEL ldn

The PLABEL function will print the contents of the label of the removable volume mounted on the specified logical device. If the volume is a scratch volume, an appropriate message will be printed. Otherwise, the following volume information will be printed:

1. volume name

2. device type and subtype

3. volume set to which the volume belongs

4. master volume information.

> PDTRACK ldn

The PDTRACK function causes a formatted listing of the defective tracks table for ldn to be printed.

> PFSPACE ldn

The PFSPACE function causes a formatted listing of the free space table for ldn to be printed.

> DTRACK ldn

The DTRACK function will allow the user to process defective track errors which may have been detected during normal access to to the volume or during the INIT function as described above. An error encountered during normal access to the device will cause the accessing program to be aborted; the operator can later, through the DTRACK function, assign an alternate for the defective track.

The detective will cause the defective tracks Table for the device to be examined. For each each defective track noted in the table, the user will be interrogated as to the disposition of the suspected track. The dialog and dispositions available are similar to those provided by INITIAL when suspect tracks are found on system domain devices. If a data track is deleted or reassigned then all files having extents on the track will be purged. In this instance the master volume of the set would have to be mounted.

> EXIT

The EXIT function allows the user to exit from the subsystem.

VOLUME BACKUP

4.1 Backup Mechanisms

The backing-up of files residing on a volume set can be accomplished in two ways. First, files from a home volume set can be stored and retrieved from tape by use of the :STORE and :RESTORE commands. In this way a subset of the files which might be contained on the set and which belong to a particular group or account associated with the volume set can be permanently backed-up.

Secondly, a single volume can be copied to another volume of the same type through the COPY function of the VINIT subsystem. A user of the VINIT subsytem must have operator capability. The COPY function is described in sub-section 4.6.

4.2 Generation Protection

The Private Volumes facility will provide a mechanism to assure that all the volumes of a multi-volume volume set are of the same generation, i.e., that they were all copied from members of the same physical volume set. The facility will also allow the user to specify explicitly which copy of a volume set she desires to access. Without such protection mechanisms it be possible to either intermingle the members of a volume set created at different times by COPY or otherwise allow an incorrect version of a set to be mounted.

When a volume is copied or initialized, a generation index will be placed in the label of the newly created volume. This index will be examined by the system on a subsequent mount of the set. The actual value to be assigned is at the control of the user and can be specified by the GEN parameter of both the INIT and COPY functions of the VINIT subsystem.

The generation indices of the member volumes will be checked when a volume set is mounted. Each member volume must have the same index as that of the master volume of the set; if a generation index was specified during an explicit mount, the index of the master volume must match that specified. If there is a mismath of indices, an appropriate message will appear on the console.

USER AND CONSOLE COMMANDS

.1 ser Commands

.1.1 New User Commands

.1.1.1 Volume Set Maintenance

1. :NEWVSET (described in sec. 4.3.4.2)

2. :ALTVSET (described in sec. 4.3.4.3)

3. :PURGEVSET (described in sec. 4.3.4.5)

.1.1.2 Volume Set Status and Display

1. :LISTVS [vsset][,detail][;listfile]

This command will produce a formatted listing of volume set definition information.

vsset is defined to be

vsname[.group[.account] @ .group[.account] , 1 [;listfile]

@ @ @

:account

If the optional parameter is omitted, the default is the logon group's home volume set.

If the form <group>.<account> (@.group.account, @.@.account, or @.@.@) is used, then the home volume set(s) of the designated group(s) are displayed.

This command displays the following information on $STDLIST or optionally on listfile:

1. volume set name and volume set definition creator

2. volume set member volume names and types

3. volume class names and master reference

4. ldn's & statuses of the devices on which the volume set is mounted.

A null or 0 detail specification display items 1 thu 3 where a detail of 1 will display items 1 thru 4.

2. :VSUSER [vspecifier]

This command will print the job/session numbers and users of a volume set. If the optional parameter is not specified, the users of all mounted volume sets will be displayed.

```
                 [dev
3. :DSTAT [{     .  }]
                 ALL
```

This command will display the types, statuses, and volume names for discs configured in the system. If the optional parameter is not specified, only information for user domain disc will be displayed.

If the vsnamelist parameter is omitted, then the assignments for all volume sets defined in the system are printed.

The default for listfile is $STDLIST.

8.1.2 Changes to Existing MPE User Commands

Certain of the currently existing MPE user commands will change to accomodate private volumes. The changes required are either syntax modifications or changes in the meanings of the commands. The commands which follow will require such changes; only the changes themselves are described.

8.1.2.1 File Handling Command Changes

1. :BUILD filereference [;DEV = device]

Specification of a device parameter will cause, like FOPEN, a user initiated mount to occcur. However, unlike FOPEN, the volume set will remain mounted until the user enters :DISMOUNT or the job or session terminates.

2. :RENAME oldfileref, newfileref

If the group and account designations given in the old and new file references are the same, the file is renamed. If the groups are different and their respective home volume set are different, the command will fail.

```
              $OLDPASS, newfileref
3. :SAVE [                        ]
         .    tempfileref
```

If the target group is different from the FOPEN/:BUILD group, and their respective home volume sets are different, the command will fail.

3.1.2.2 File Maintenance Commands

File maintenance commands require that a specified file's home volume set has been previously mounted. These commands will not cause implict mounts to be invoked.

1. :LISTF filesetlist

A file will not be listed unless its home volume set is mounted.

2. :RESTORE tapefile [;filesetlist] [;DEV = device]

The DEV parameter in this command is syntactically identical to the DEV parameter of the FOPEN intrinsic, and the :FILE and :BUILD commands. This command will operate as specified for the current version of MPE with the following exceptions:

a. Any file belonging to a group whose home volume set is not mounted will not be restored.

b. If the DEV parameter is specified, the files will be restored on the logical device or device class formed by the intersection of the specified parameter and the mounted home volume set.

c. If the DEV parameter is not specified, an attempt will be made to:

1). First, restore the file on the logical devices formed by the intersection of the type/sub-type specified in the file's file label and those of the mounted home volume set.

2). Secondly, if the above intersection is null, an attempt is made to restore the file on the device class formed by the intersection of the device class specified in the file's file label and that of the mounted home volume set.

d. If for any reason the intersections described above are null, the system will attempt to restore the file on any member of the mounted home volume set.

3. :STORE [filesetlist]

This command will store only those files whose home volume set is mounted.

STORE and RESTORE operations will cause the logon group's home volume set to be implicitly mounted due to the opening of STORE and RESTORE work disc files. Due to this anomaly, files destined to or originating from the logon group will be successfully stored and restored without first mounting their

home volume set. All other files will require that their home volume set first be mounted for the STORE/RESTORE operation to succeed.

8.1.2.3 File Security Handling Commands

1. :ALTSEC fileref

2. :RELEASE fileref

3. :SECURE fileref

These file security commands will fail if the group's home volume set is not mounted.

8.1.2.4 Accounting Structure Commands

For the following commands, the volume set, volset, need only be mounted if the SPAN subparameter is to be specified.

1. :NEWACCT acctname [;VS=volset :SPAN]

2. :ALTACCT acctname [;VS=volset :SPAN]

3. :PURGEACCT acctname [;VS=volset]

If groups still remain (were in-use) within acctname after this operation, then the command is terminated at this point; otherwise the entire account is deleted from the specified volume set.

4. :NEWGROUP groupname [;VS= [volset [:SPAN]]] (Sec. 3.5.2)

5. :ALTGROUP groupname [;VS= [volset [:SPAN]]] (Sec. 3.5.2)

If volset is different from the old volume set, and the old volume set is the system set, then the old volume set must be examined for the presence of files belonging to groupname. If no such files are found, the command will succeed, and the group is reassigned to volset as its new home volume set.

A null volume set specification to the VS= parameter will cause the group to be reassigned to the system volume set. If the old volume set is the system volume set and contains files belonging to groupname, the command will fail, i.e., nothing is altered.

It is permissable to reassign a group to a different volume set regardless of presence of files belonging to groupname so long as the old volume set is not the system set and that group groupname is not currently bound (either explictly via the MOUNT command or implicitly via FOPEN) to its home volume set.

6. :PURGEGROUP groupname [;VS=volset]

The group groupname will be purged from the specified volume set only if the volume set has been first mounted.

7. :LISTUSER username

8. :LISTGROUP groupname

9. :LISTACCT acctname

These commands are extended in that they will return private volume information, e.g. VS names. No volume sets need be mounted for the operation of this command.

4.8.1.3 Miscellaneous Commands

4.2 Console Commands

Private Volumes will add the following console commands to the system.

1. =DSTAT {{ ldev / ALL }}

(Same as user command :DSTAT except that parameter is required)

2. =VSUSER

(Same as user command :VSUSER)

3. =V4OUNT

(Discussed above in Section 4.4.2.1)

4. =MOUNT

(Discussed above in Section 4.1.2.2)

5. =DISMOUNT

(Discussed above in Section 4.4.2.2)

RESOURCE ACCOUNTING

The resources pertinent to Private Volumes for which a user may be charged are

a. Volume disc space usage

b. mount time (spindle usage).

Disc space accounting will be performed in the same manner as on non-private volume systems: groups and accounts will be charged for the sectors used. However, under Private Volumes, the space charged will be that allocated on the home volume set which may or may not be a private volume set.

Users may also be charged for the time that a volume set was mounted for their use. The system log will provide the times that a particular volume set was logically or physically mounted/dismounted for a particular user.

10 SYSTEM LOG

The Private Volumes facility will cause two additional members to be added to the set of system log entry types: the volume mount/dismount entry and the volume set mount/dismount entry.

10.1 Volume Mount/Dismount Entry

This entry will record the time that a removable volume was physically mounted on, or dismounted from, a non-system domain drive. Additional volume related information will be logged also. The system, after being cold-loaded, will interrogate the devices in the non-system domain to determine which have volumes mounted; volume mount entries will be created for all devices having mounted volumes. The format of the log entry is:

```
words 0 -5: standard log entry heading
word 6.(14:2) = 0 - mount
                1 - dismount
                2 - new volume created by >init function
      .(13:1) = 1 - volume mounted at cold-load time
      .( 8:4) = volume type/status
                0 - valid Private Volume
                1 - SCRATCH volume
                2 - system volume
                3 - unformatted volume
                4 - unreadable volume
                5 - serial disc volume
      .( 0:8) = volume table index
word 7.( 8:8) = logical device number
      .( 4:4) = device subtype
words  8 - 11: volume name (8 bytes)
words 12 - 15: vs name (8 bytes)
words 16 - 19: vs group (8 bytes)
words 20 - 23: vs account (8 bytes)
```

10.2 Volume Set Mount/Dismount Entry

This entry will record the time of mount/dismount as well as volume set definition information pertinent to a volume set that was mounted or dismounted by either a user- or operator-initiated request. A log entry will be made irrespective of the number of other users that may be accessing the set. The format of the entry is:

```
words  0 - 5: standard log entry heading
word 6.(15:1) = 0 - mount
                1 - dismount
      .(12:3) = type of request:
                0 - :MOUNT/:DISMOUNT with directory binding
                1 - :MOUNT/:DISMOUNT without directory binding
                2 - unconditional implicit mount
                3 - conditional implicit mount
```

```
                    4 - =MOUNT/=DISMOUNT
                    5 - dismount due to job/session termination
word 7.( 8:8) = pin number of accessor
      .( 0:8) = number of accessors (including
                this accessor)
words  8 - 11: user name (8 bytes)
words 12 - 15: group name (8 bytes)
words 16 - 19: account name (8 bytes)
words 20 - 23: vs name (8 bytes)
words 24 - 27: vs group (8 bytes)
words 28 - 31: vs account (8 bytes)
word 32 = number of volumes mounted or dismounted
word 33.(0:8) = first volume logical device number
       .(8:8) = first volume device subtype
word 34.(0:8) = second volume logical device number
       .(8:8) = second volume device subtype
word 35.(0:8) = third volume logical device number
       .(8:8) = third volume device subtype
word 36.(0:8) = fourth volume logical device number
       .(8:8) = fourth volume device subtype
word 37.(0:8) = fifth volume logical device number
       .(8:8) = fifth volume device subtype
word 38.(0:8) = sixth volume logical device number
       .(8:8) = sixth volume device subtype
word 39.(0:8) = seventh volume logical device number
       .(8:8) = seventh volume device subtype
word 40.(0:8) = eighth volume logical device number
       .(8:8) = eighth volume device subtype
```

11 DISC ERROR RECOVERY

Two classes of errors may occur on non-system domain devices: track-specific errors and drive errors. Recovery procedures will differ according to the type of error detected.

11.1 Track-Specific Error

Track-specific errors are those which the controller determines to be the result of a failure on one track on a volume. The procedures available for the recovery of a track-specific error will depend on the type of track found defective. Tracks may be either data tracks which contain only file-related information or system tracks which contain information used by the system to control access to the volume.

11.1.1 Data Track Errors

The occurance of a track error on a data track will cause an error condition to be returned to the accessing program. Also, the Defective Tracks Table for the volume would be updated to reflect the presence of the suspect track and the following console error message will be printed:

I/O/<time>/ DATA TRACK ERROR ON LDEV # ldn

The operator should enter an =DISMOUNT command to prevent further access to the volume set and then invoke the VISIT subsystem in order to dispose of the defective track on the volume. A data track error will not affect other users of the volume set unless they attempt to access the suspect track.

11.1.2 System Track Errors

The occurance of a track error on a system track will cause the following message to be printed on the console:

I/O/<time>/ SYSTEM TRACK ERROR ON LDEV # ldn

11.2 Drive Errors

All disc errors occuring on a non-system domain device which are not track specific or which are found to occur on more than one track will be referred to by the generic term of drive error. Because of the ambiguity inherent in the error codes returned by the disc controllers upon detecting errors, a drive error may actually be the result of a media (track) failure, a [illegible] head crash. The term drive error is used to refer to all such possible conditions.

In the occurance of a drive error on a non-system domain device, the operator will be notified by the following message:

I/O/<time>/ DRIVE ERROR errornum ON LDEV # lgn

where errornum is an integer indicating the error code returned by the disc controller. Recovery procedures following a drive error will be the same for all values of errornum; errornum is provided for informational purposes only.

Once a drive error has occured on a device, all subsequent I/O operations to the device will return an error condition to the caller; the caller will determine what actions he should take in such a case. Other users of the volumes set will not be affected by the error unless they attempt to access the detective volume/drive.

It is possible that a drive error condition may be corrected by placing the suspect volume on a different drive. If the volume can be successfully mounted, i.e. a volume mounted message appears on the console, then it is possible that the error was not a media error and the volume can be accessed on the new drive. In order to attempt such a recovery, the operator would perform the following operations:

a. =DISMOUNT the volume set having the suspect volume

b. EXAMINE THE VOLUME FOR SIGNS OF A HEAD CRASH. A volume subject to a head crash on one drive may itself cause a head crash if moved to another drive.

c. Place the suspect volume on a different drive and note the the console message which appears. If the message indicates that the volume was successfully mounted, the operator should proceed to step d. below. If the volume cannot be successfully mounted, either the "unreadable label", or "unformatted volume" message may occur; in either case no further recovery steps can be taken.

d. Re-initiate the jobs who were using the set at the time of the drive error.

LOGGING

IDENTIFICATION

.1 Name

MPE IIB USER LOGGING

.2 Abstract

MPE IIB User Logging provides a flexible new logging capability for MPE IIB users. The facility supports both system and private level logging. System level logging routes all log records to a common disc or tape file. Private logging dedicates a disc or tape file to a particular user.

DESIGN CONSIDERATIONS

.1 Hardware Environment

USER LOGGING is an integral part of MPE IIB. It will run with any of the supported software on the system. Logging over DS lines has not been considered in the project.

.2 Software Environment

USER LOGGING will run on any hardware that runs MPE IIB. If logging to tape is desired, the hardware configuration must have a tape drive.

DESCRIPTION

.1 Overview of Operation

MPE IIB user logging provides an important new facility for MPE IIB users. It allows users and subsystems to journalize additions and modifications to MPE IIB and subsystems files, and provides a mechanism whereby the journal entries can be used to recover these files in the event that it becomes necessary.

MPE IIB logging provides the user with two unique types of logging. For the average user, there is the system logging facility which takes all log records and writes them to one central logging file. The logging identifier as described in this document is used to distinguish ownership of the individual log records for the recover utility. This type of logging has the advantage of making both tape and disc files sharable resources. Thus the facility improves the useability of an important system resource while logging.

The other type of logging provided by the facility is private logging. This method can be used when it is necessary for the application to maintain it's own transaction records for

reasons of recovery, auditing, or security. With this type of logging, all predetermined logging identifiers that have been assigned to a private logging process are separated by the logging facility and written to a private disc or tape file.

Both private and system logging operate in the same manner so the description that follows applies to both.

The MPE IIB logging facility consist of a logging buffer file, a logging process, a logging data segment, and an optional logging tape file for each logging process. The logging buffer file has two functions in the system. First, if the operator requests logging to tape, the logging buffer file buffers I/O between the user's process and the logging tape file. This will reduce the amount of overhead due to I/O to the logging tape. It also provides for more "even" response to interactive users who won't "feel" the I/O to tape taking place. If the operator specifies logging to disc, the logging buffer assumes the role of the logging file. When this occurs, all records received by the logging process will remain in the logging buffer. The file will expand, within limits, as the need for more space occurs.

The logging process acts as the interface between the logging buffer file and the logging tape file when logging to tape is requested by the operator. This process runs independent of the user's process and thus the logging process can be writing records to tape at the same time that users are adding additional records to the logging buffer. Again, this aids in making the users response from the logging facility quick and even. When logging to disc is specified, the logging process function changes. In this mode of operation, the process insures that enough disc space is available so that logging can continue. It does this by opening additional logging files when the current file reaches a predetermined point. Thus, when logging to disc, users should rarely be impeded while more disc space is being made available.

The logging data segment is divided into two areas. First there is the buffer area which is used to buffer I/O between user's processes and the logging buffer. Secondly, there is the communications area in which users and the logging process send messages. These messages generally pertain to the status of the users process or state of the logging process.

Users accessing the log buffer file do so by placing their records in the buffer area of the logging data segment via intrinsics. The process that fills the logging buffer area of the data segment writes it's contents to the log buffer file. It then checks the communications area of the logging data segment to insure that the logging process is active. If the logging process is sleep, it activates the logging process.

In the event that a logging buffer becomes full, all users accessing it will be suspended. The communications area will be used to indicate that they are waiting to access the logging buffer. Once the logging process makes space available in the buffer, it will interrogate the communications area and activate all user processes in the wait state. When the logging buffer has been completely emptied, the logging process will sleep.

When logging to disc is specified by the operator, the logging process will write log records to a disc file instead of a tape file. All other considerations described above are the same.

Access to the logging facility is controlled by logging identifiers which are created by users having the new logging capability. The creator of a given logging identifier can allow other users access to the logging facility ,either private or system, by giving them the logging identifier and password, if any, associated with it. These users must use this information in the OPENLOG intrinsic to obtain access to the logging system.

3.2 Features

1) Simple user interface

2) Ability to log to disc or tape

3) ANSI Standard Labels used to protect the Log file when on tape

4) Most MPE IIB supported languages can use the facility

5) Private and System level logging

3.3 Goals

1) Provide a method whereby MPE IIB users can log data relevant to the recovery of MPE IIB files.

2) Provide a method of retrieving information from the logging System.

3) Provide for a secure Logging System so that every user's data is protected.

4) Provide a clean, simple interface for the above.

USER INTRINSICS

The following procedures constitute the users programmatic interface to the logging system.

```
PROCEDURE OPENLOG(INDEX,LOGID,PASS,MAXREC,MODE,STATUS);
DOUBLE INDEX;
INTEGER MAXREC,MODE,STATUS;
ARRAY LOGID,PASS;
OPTION EXTERNAL;
```

The OPENLOG intrinsic is used to obtain access to the logging system. It insures that the caller has access to the logging facility by checking the validity of the logging identifier and password.

PARAMETERS:

INDEX- An integer returned to identify the logging access in other calls by the user

LOGID- An array in which the user supplies his logging identification. It can be up to eight characters. The identifier is stamped on every record written by the user and is used by the recovery utilities to identify ownership of the records during recovery.

PASS - An array in which the user supplies the password associated with the logging identifier.

MAXREC- An integer in which the user supplies the maximum record length to be passed to the logging system via the LOGWRITE intrinsic. The parameter is used to optimize the size of the logging system buffers and data segments.

MODE- An integer in which the user indicates whether or not his process should be suspended in the event that his access to the logging facility cannot be obtained immediately. The only reason that this would occur is when the user is sharing a logging buffer which is currently full and the logging process has been to busy to empty it.

STATUS- An integer that the logging system uses to return error information to the user. Fatal errors are indicated by a negative number, warnings or non fatal errors are positive. OK status is identified by zero.

```
PROCEDURE WRITELOG(INDEX,DATA,LEN,MODE,STATUS);
DOUBLE INDEX;
INTEGER LEN,MODE,STATUS;
ARRAY DATA;
```

The WRITELOG intrinsic is used to write a physical record to a logging file. When a user calls the WRITELOG intrinsic it, provided there is room, places the requested transfer into the buffer area of the logging data segment. If this request fills the buffer, the intrinsic will write the entire buffer area in the logging data segment to the logging buffer on disc. If the logging buffer is full, the intrinsic will interrogate the communications area of the logging data segment to insure that the logging process is active. If the logging process is active, the intrinsic will suspend the caller and notify the logging process that it is waiting by making an entry in the communications area of the logging data segment. If the intrinsic finds that the logging process is sleep, it will activate the logging process prior to sleeping the caller. If the user specifies in the MODE parameter that he does not want to be impeded, the intrinsic will under no circumstances sleep the caller. Instead, the caller will be notified in the STATUS parameter that his request was not completed because the logging buffer was full. In this case, it is the responsibility of the user to resubmit the request at a latter time.

PARAMETERS:

DATA- An array in which is passed the actual information to be logged.

LEN- The length of the data in DATA. A positive count indicates words, and negative count indicates bytes.

INDEX- The parameter returned from LOGOPEN that identifies the use's access to the logging system.

STATUS- An integer that the logging system uses to return error information to the user. Fatal errors are indicated by a negative number, warnings or non fatal errors are positive. OK status is identified by zero.

MODE- An integer which specifies whether the user wants his process impeded by the logging process in the event that the logging buffer becomes full. If set, the WRITELOG intrinsic will, in the event that it is not possible to complete the request without

impeding the process, return an indication in the status word that the request was not completed.

```
PROCEDURE CLOSELOG(INDEX,MODE,STATUS);
DOUBLE INDEX;
INTEGER MODE,STATUS;
OPTION EXTERNAL;.
```

The CLOSELOG intrinsic is used to cease access to the logging system. When this intrinsic is called, the logging entry in the logging data segment is deleted. This closes access to the caller. The intrinsic also writes a record to the logging system specifying that this particular caller has closed his logging file. The writing of the record to the logging system is subject to the same considerations as in the WRITELOG intrinsic. In otherwords, the process can be impeded if the logging buffer is full.

PARAMETERS:

INDEX- the parameter returned from LOGOPEN that identifies the user's access to the logging system.

MODE- An integer in which the user indicates whether his process should be suspended in the event that his access to the logging facility cannot be obtained immediately. The only reason that this would occur is when the user is sharing a logging buffer which is currently full and the logging process has been to busy to empty it.

STATUS- An integer that the logging system uses to return error information to the user. Fatal errors are indicated by a negative number, warnings or non fatal errors are positive. OK status is identified by zero.

```
PROCEDURE LOGINFO(INDEX,LOGSTAT,STATUS);
DOUBLE INDEX;
INTEGER LOGNO,STATUS;
ARRAY LOGSTAT;
OPTION EXTERNAL;
```

The LOGINFO intrinsic is used to programmatically obtain information on the status of the logging process and certain information on the user who calls the intrinsic. The intrinsic interrogates the logging communication area to get the items requested.

PARAMETERS:

INDEX- The parameter returned from LOGOPEN that identities the user's access to the logging system.

STATUS- An integer that the logging system uses to return error information to the user. Fatal errors are indicated by a negative number, warnings or non fatal errors are positive. OK status is identified by zero.

LOGSTAT- An array in which information about the status of the logging process is returned. This array will be formatted with each entry returning a particular item relating to the logging process or logging user. The actual contents of the array will be defined in the project however, infomation pertaining to the activity or inactivity of the logging process, whether it is executing or not, whether system or private log, and whether the logging buffer is full or not are sure to be included.

USER COMMANDS

:GETLOG logid [;PASS=password] [;LOG=logfile [,DISC | ,TAPE]

The GETLOG command is used to establish a logging identifier (LOGID) on the system. The creator of the logging identifier can allow other users access to his logging identifier by notifying them of the logging identifier and password. Users accessing the logging system with this identifier must supply the identifier and the password in the LOGOPEN intrinsic.

PARAMETERS:

logid - The logging identifier to be established on the system. This should be a string of characters (up to eight) that are meaningful to the user or his application.

password- The password to be associated with the logging identifier. This parameter protects the user from illegal use of his identifier.

logfile- The destination file name for the logfile. This parameter is used when the user wants to establish a "private" logging file. If this parameter is omitted, the destination file becomes the system logging file.

:RELLOG logid

The RELLOG command is used to remove a logging identifier from the system. The command can only be issued by the user who created the logging identifier. After issuing this command, programs containing this logging identifier will not be allowed to access the logging system.

PARAMETER:

logid - The logging identifier to be removed from the system.

:LISTLOGID [logid]

The LISTLOG command is used to list the currently active logging identifiers on the system.

PARAMETER:

logid - An optional parameter that is used when the user wants to verify the existence of a particular logging identifiers. If this parameter is omitted, all currently active logging identifiers will be listed.

:ALTLOG logid [;PASS=password] [;LOG=logfile [,DISC] | [,TAPE]

The ALTLOG command is used to alter the attributes of an existing logging identifier. The command can only be envoked by the creator of the logging identifier.

PARAMETERS:

logid - The logging identifier who's attributes are to be changed.

password- This optional parameter indicates that the password for the logging identifier is to be changed or modified.

logfile- The destination file name for the logfile. This parameter is used when the user wants to establish a "private" logging file. If this parameter is omitted, the destination file becomes the system logging file.

:NEWACCT acctname,username [;CAP=LG]

The NEWACCT command has been modified to include the new logging capability (CAP=LG). The account manager can assign this capability to any other users in the account. All other parameters of the command remain the same.

:ALTACCT acctname [;CAP=LG]

The ALTACCT command has been modified to include the new logging capability (CAP=LG). All other parameters of the command remain the same.

:NEWUSER username [;CAP=LG]

The NEWUSER command has been modified to include the new logging capability (CAP=LG).

:ALTUSER username [;CAP=LG].

The ALTUSER command has been modified to include the new logging capability.

:SHOWLOGSTATUS

The SHOWLOGSTATUS command is used to obtain information about the status of the currently opened logging files.

CONSOLE COMMANDS

LOG [logid] [START]
[RESTART]
[STOP]

The LOG command is used to control both system and private logging processes.

Parmeters:

logid - An optional parameter which is used to indicate first that a private logging system is to be initiated and, second the logging identifier that will go to that process.

START - This parameter specifies that a logging process, either system or private, is to be initiated.

RESTART - This parameter specifies that a system or private logging process is to be restarted following a system restart (COLD/COOL/WARM).

STOP - This parameter specifies that a system or private logging process is to be terminated. This action may not be immediate. The command does not take effect until all users currently accessing a logging facility have closed their logging files. However, all new request to access the logging process will be denied.

7 RECOVERY UTILITY

The recovery utilty provides the important link between logging and recovering data. While the utility does not provide for automatic recovery, it does aid the user in obtaining his log records so that he can perform recovery on his own behalf. The utilty separates system log tapes by logging identifier and creates a disc based file on which the users recover program can run to complete recovery.

7.1 RECOVER

The RECOVER command is used to initiate the recovery utility. The utility will prompt the user with ">" to indicate that it is ready to accept input. Input is in the form of the logging identifier to be recovered, and the filename that records having that logging identifier are to written to. The named file must be in the domain of the user that created the logging identifier. If recovery is taking place against a private logging file, the user who issues this command must be the creator of the logging identifier. In the case of a system logging tape, the user issuing this command must be the system manager.

7.2 Example

```
 RECOVER

 ENTER LOGID,RECOVER FILE NAME

>NEAL,RFILE.MACK.DEE
>USER,URFILE.UGROUP.UACCT
>CR

 ALL FILE RECOVERED

 END OF SUBSYSTEM
```

In the example above, the user recovering logging identifiers NEAL and USER must have created both. If the logging identifiers belong to a system logging process, the user who issues this command must be the system manager.

7.3 SYSTEM PROGENITOR

The new system progenitor plays an important part in the logging facility. A new function for the progenitor will be to check the logging buffer file and determine if logging to tape was taking place prior to a restart (WARM/COOL/COLD). If this condition is found to be true, the progenitor will ask the operator to identify the physical drive that the logging tape is mounted on and to insure that the drive is on line. It is important that the operator does not rewind the tape as

this will cause the logging facility to search through the entire contents of the tape to reach the point where the crash occured. If the tape is not rewound, the logging recovery mechanism in the progenitor will back space the logging tape one block and read the last block written. It will then compare the record number of the last block written with those of the records that remain in the logging buffer. Once the next record to be written has been found, the mechanism will flush the logging buffer to the logging tape. A special record will be written to indicate the point where the crash occured. After doing this, the file will be rewound and made available to the logging process to continue logging.

DEFINITIONS

logid

A logging identifier. It is up to eight characters long and serves to identify a logical logging process.

LOGGING PROCESS

An MPE IIB process which either writes logging records to a tape or disc file.

SYSTEM LOGGING PROCESS

An MPE IIB logging process which can contain more than one logical logging process. For example, more than one logging identifier can be written to this process.

PRIVATE LOGGING PROCESS

An MPE IIB logging process which is dedicated to one logical logging identifier (LOGID).

LOGGING DATA SEGMENT

An MPE IIB data segment that is used to buffer I/O between logging users and the logging buffer file on disc. It is also used for interprocess communication among logging users. There is one logging data segment for each logging process.

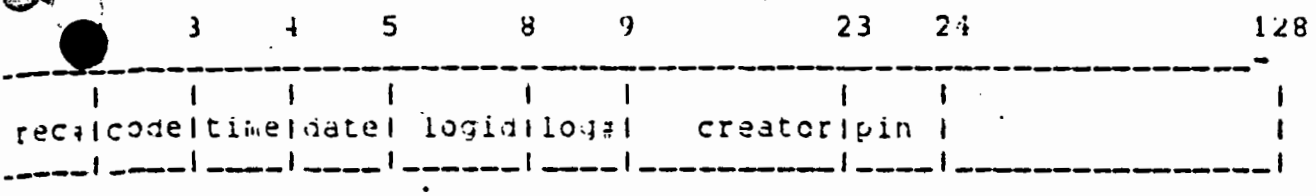
LOGGING BUFFER

An MPE IIB file that is used to buffer I/O between a logging process and a logging tape file. There is one logging buffer for each logging process.

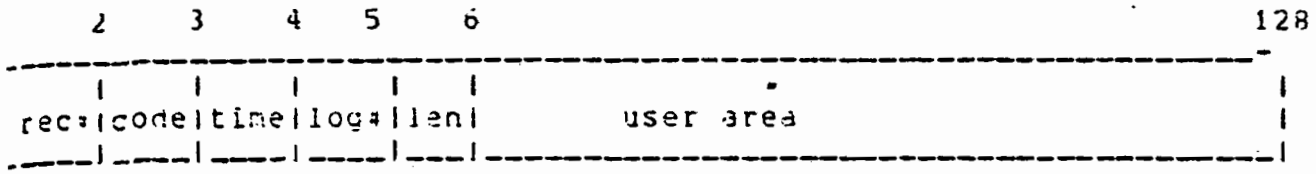
LOG FILE

An MPE IIB file which holds log records. The file may be on disc or tape.

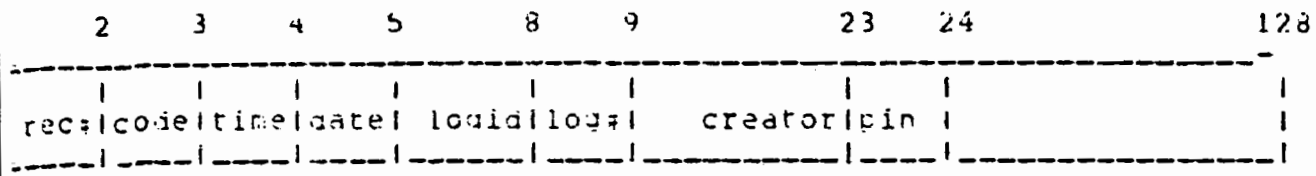
LOG RECORD AT OPENLOG



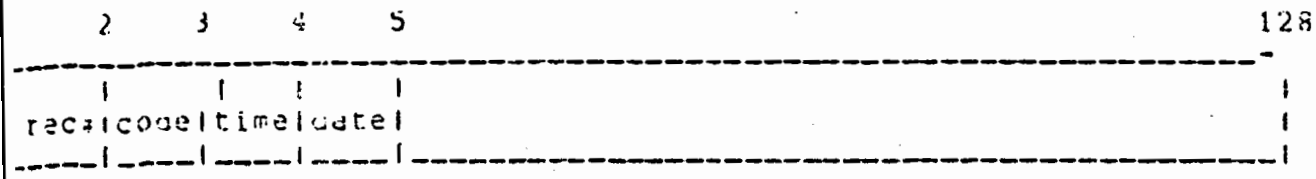
USER OR SUBSYSTEM LOG RECORD



LOG RECORD AT CLOSELOG



CRASH MARK



CODE DEFINITION (TEMPORARY)

CODE=1 > OPEN LOG 2 > USER/SUBSYSTEM RECORD 3 > CLOSE LOG 9 > CRASH MARKER

SERIAL DISC INTERFACE

IDENTIFICATION

.1 Name

SERIAL DISC INTERFACE

.2 Abstract

To provide fast backup/recovery to a "private disc", the serial disc interface will allow non-system-domain discs to be treated as serial devices.

DESIGN CONSIDERATIONS

.1 Hardware Environment

Any Series II or successor capable of running MPE II or MPE IIB with at least one non-system-domain disc.

.2 Software Environment

MPE IIB with PRIVATE VOLUMES. The serial interface will be developed on the Series II software base.

.3 Performance / Customer Satisfaction

The serial disc interface should have little effect on the overall system performance. However, for single jobs requiring large amounts of serial data storage, such as a full SYSDUMP, there should be a significant positive impact on the run time for that job. This speed increase will be gained by the increased data transfer rate of the 7905/7920 discs over the 7970 tape drive and the savings in time necessary to mount continuation reels due to the storage capacity of the 7905/7920 discs.

The convenience of another serial storage medium with a range of capacities from .5 to 50 MEGABYTES will contribute to the customer's overall satisfaction with the HP3000.

DESCRIPTION / USER INTERFACE

.1 Comparison to Magnetic Tape

All user programs designed to use the magnetic tape storage medium should run using the serial disc interface without any modification.

3.2 Privileged Mode Users Only

There is a feature of the serial disc interface for privileged mode use ONLY which guarantees that data written will not span any disc defects. This also ensures that data written can be read by an SIO program. Data written in this mode, however, CANNOT be read by the serial disc interface mechanism.

This special feature is enabled and disabled by the control field of the FWRITE intrinsic. A control code of %1001 will enable the "CONTIGUOUS WRITE" feature and a control code of %2001 will end the contiguous block and disable the feature. In the event two unrelated contiguous blocks need to be written without any intervening data, the control code %1001 will end the current block and begin the next. The write end-of-file command of FCONTROL will also imply an end to the current contiguous block. It is advantageous to keep the blocks as small as possible. Relocation of a large block around a disc defect wastes disc space and increases system overhead.

THE CONTIGUOUS BLOCKS ARE TRANSPARENT TO THE SERIAL DISC INTERFACE. An attempt to read from the serial disc using the serial interface when contiguous blocks were written will result in only those records that were written WITHOUT the %1001 option.

EXAMPLE:

Suppose this serial disc exists-

Records 1-3 were written with the contiguous option OFF
Records 4-6 were written with the contiguous option ON
Records 7-9 were written with the contiguous option OFF

If this disc is read using the serial disc interface, the fourth read request will not return record number four, but record number seven.

The data written with the contiguous feature on can be read by a direct ATTACHIO call to the serial disc unit. The relocation of contiguous blocks around defective tracks, however, makes it impossible to precetermine the starting address of a contiguous block. To allow the privileged user to gain access to this information, an uncallable procedure (FINDSDISCGAP) will be provided. Full documentation on the calling procedures and capabilities of FINDSDISCGAP are provided with the module. To read a contiguous block, use this procedure to determine its address and length.

```
INTEGER PROCEDURE FINDSDISCGAP(LDEV,BLOCK,ADR,LEN);
VALUE LDEV,BLOCK;
INTEGER LDEV,BLOCK;
DOUBLE ADR,LEN;
OPTION VARIABLE,EXTERNAL;
```

LDEV -Logical device number of serial disc unit
BLOCK-A positive value corresponding to the number of the contiguous block sought.

i.e. 5 will return the starting address of the 5th contiguous block.

ADR -Sector address of start of contiguous block (Returned Value-REQUIRED)

LEN -Sector length of contiguous block (Returned Value-OPTIONAL)

FINDSDISCGAP -Serial disc errorcode (See SIERR module for a full list of errorcodes).

TE: A serial disc file must be open to the logical device at the time of te call for FINDSDISCGAP to return a valid address and length.

OPERATOR INTERFACE

.1 Operator Control

A user's request for a serial disc will be relayed as a console message similar to an IPL tape request. The operator will be required to respond with the logical device number of the drive and whether a "write ring" is to be considered present or not. The LDEV only will result in no write ring. The "write ring bit" will reside in the serial disc's data segment.

SDSTAT will be extended to display the status of serial discs also.

When a serial disc file is closed, the serial interface will inform the operator that the drive is now available for other use by a console message.

A message will be output to the console whenever a serial disc is physically mounted on a disc drive.

.2 Use as a Cold Load Medium

The serial disc will be supported as an alternative to magnetic tape for SYSDUMP output. Use of this option will not effect the operation of SYSDUMP in any way.

To perform a cold load on the HP3000 Series II from a serial disc generated by SYSDUMP:

1) Press the RUN-HALT switch if the computer is running.
2) Mount the serial disc back on one of the NON-SYSTEM-DOMAIN drives and switch it on.
3) Set the drive to unit zero and be sure no other drives on the same controller are also set to unit zero.
4) Wait for the drive to become ready.
5) Set bits 0-7 of the SWITCH REGISTER to zero.
6) Set bits 8-15 of the SWITCH REGISTER to the DRT# of the disc drive.
7) Press the LOAD switch while holding the ENABLE switch.

8) When the computer halts, press RUN.
9) Follow the steps outlined in the INITIATOR-USER DIALOG of the SYSTEM MANAGER/SYSTEM SUPERVISOR manual for the rest of the load.

5 INTERNAL CONSIDERATIONS

5.1 Auto Recognition

The device auto recognition routine will attempt to identify any disc pack that is mounted on a drive. There will be a special mark in the disc label to identify serially initialized disc packs as such. Upon recognition of a serial disc, the auto recognition routine will set a special bit in the LOGICAL DEVICE TABLE to relay this information to the rest of MPE. A message will then be output to the console to notify the operator that a serial disc has been mounted. The serial disc structure will be compatible with the tape label software to allow recognition of a labeled serial disc.

5.2 Declaration of Serial Disc Class

Serial disc candidates are non-system-domain moving head discs. Disc subtypes may not be mixed within a single device class. Any device containing a system pack when INITIAL attempts to satisfy the VOLUME TABLE will be effectively removed from the serial disc classes. Modification to SYSDUMP and INITIAL will be necessary to facilitate recognition of serial access disc classes. (CLASS CHANGES of I/O CONFIGURATION CHANGES of SYSDUMP and INITIAL will be changed to include the following question during disc class definition.

SERIAL DISC CLASS? (YES/NO)

5.3 Reference to Serial Discs by Class or Ldev

FOR CLASS REQUESTS:
The FOPEN call to the FILE SYSTEM will cause interrogation of the class access type field of the DEVICE CLASS TABLE. Recognition of the serial disc class access type will cause an OPERATOR REQUEST for the LOGICAL DEVICE NUMBER to be generated. After it has been verified that the LDEV belongs to the requested class, sequence will proceed as an LDEV request.

FOR LDEV REQUESTS:
The call to FOPEN will interrogate the VOLUME TABLE to insure that the device is not in the RANDOM ACCESS DOMAIN. If the device is not a random access device, the serial disc mounted bit of the LPDT will be tested. FALSE will result in a DEVICE NOT READY message and a return to the LPDT test. TRUE will allocate the device exclusively to the user. FOPEN will then change the disc device type to a special type (%37) and relay that to the ACCESS CONTROL BLOCK. From this point, the file system checks device type from the ACB rather than the LDT.

SYSDUMP and INITIAL will be modified to not allow configuration of devices with this special (%37) type.

5.4 Disc Label

The disc label will be consistent with any auto recognition and private volume initialization restrictions. Modifications will be made to the private volume initialization routines to provide for the initialization of serial access discs through the SERIAL command.

5.5 Defective Tracks Table

The serial interface will access the DTT on a read-only basis. It will rely on the disc driver to maintain the table.

5.6 Data Storage / Record Format

The first file will begin in sector zero of track one. Each FILE of the serial disc will begin on a sector boundary, but the individual records within the file will be written without regard to sector boundaries. Each record will have its record length stored as a positive byte count in the word immediately before and after the record information.
For example:
/reclen/RECORD/reclen/reclen/RECORD/reclen/reclen/RE...

Note for privileged users-
Each RECORD of a contiguous block will begin on a sector boundary.

The records will be buffered to allow multiple-sector transfers to the disc units. The length of this buffer will be a constant value for all serial disc I/O.

An EOF mark will consist of zeroes for the remainder of the current sector.

5.7 End of Tape

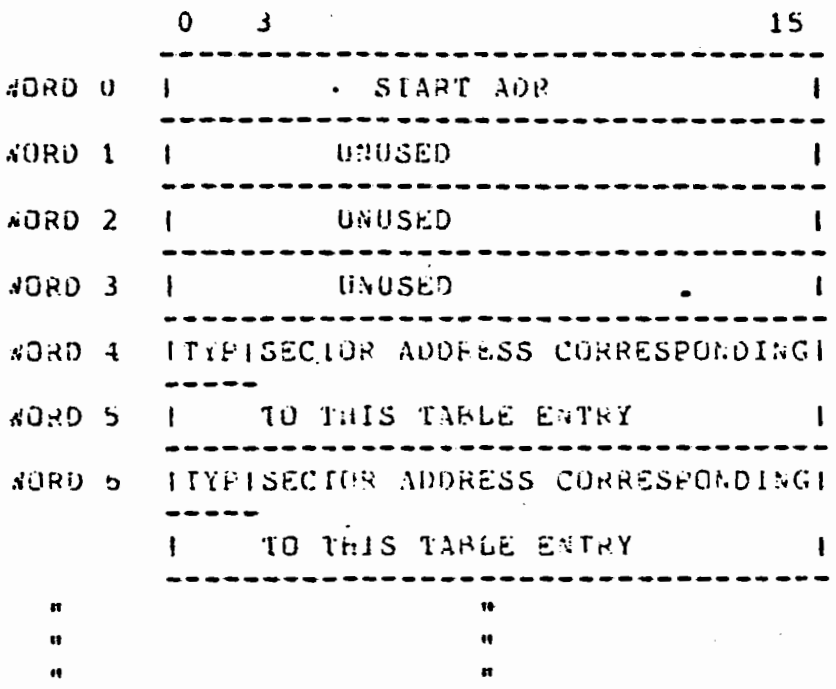
When any record extends into the last track of the serial disc, the EOT condition is returned by the serial interface. As with magnetic tape, it will be the responsibility of the programmer to avoid running off the end of the storage medium.

5.8 Device Directory (DDD) Management

A dynamic chain of serial discs similar to the class chain in the DDD is required. An entry will be added upon device allocation and deleted again upon deallocation. Support procedures will have to be developed and INITIAL modified to reflect this new structure.

.9 Gao Table

This table holds information pertaining to the file portion of the serial disc. It is resident to the disc, beginning in sector four.



START ADR => Sector address of sector zero of track one. This address is used to determine whether the disc is "virgin" when the device is opened.

TYP => Identifies this entry type
- 0-Sector address of EOF mark
- 1-Sector address of last record physically written on the disc Used to detect overrun on serial disc read
- 2-First sector of a "HOLE"
- 3-Last sector of a "HOLE"
- 4-First sector of a contiguous block
- 5-Last sector of a contiguous block
- 6-Undefined
- 7-End of table marker

STORE/RESTORE

1 DISCUSSION

Currently, under the present scheme of Store/Restore, all volumes of a multi-volume STORE media volume set must be scanned in order to RESTORE a file that resides on the last volume of the set. This limitation results from the manner in which the control information, the STORE file directory, is organized on the STORE media. Specifically, the file directory resides only on the first volume of the set.

Under the new scheme, each volume will carry a copy of the complete directory. The following sections will describe how this will allow for a more efficient means of RESTORE'ing selected files from multi-volume STORE media volume sets.

2 STORE MEDIA - Current and Future

2.1 Current Media

2.1.1 Magnetic Tape

Magnetic tape is the current STORE media used. One 200 ft. reel recorded at 1600 BPI has the potential capacity of storage for 1.5 mega words of information.

2.2 Future Media

2.2.1 Serial Mode Disc Devices

a). Floppy Disc

Each Floppy will have the storage capacity of 0.5 mega words.

b). 7920 Disc (or successor)

Storage capacity exceeds 50 mega bytes.

2.3 Observation

As can be seen from the storage capacity of the current and future medias described above, the old STORE facility's limitation would require a great deal of reel/floppy/pack switching and/or scanning to accomplish selective RESTORE operations.

3 STORE DATA STRUCTURE - Old and New

Element Changes

a). Header

The Header will contain the following additional information.

1). The string "VIII" in words 14 and 15 indentifies the new data structure.

2). A checksum value in word 16 for verifying the STORE header record.

3). Word 17, if true, indicates that the first file on the volume is a partial due to volume spanning.

b). Trailer

No changes.

c). Directory

The directory is identical in format except that some file names could be flagged (bit 0 of word 0 of the 12 word directory entry).

An entry which represents the first complete file stored on a volume, with the exception of volume 1 (first file of the first volume of a set) will be flagged.

This representation allows sets of files contained on any preceding volume to be easily determined.

The first file/first volume exception noted above allows the new STORE data structure to be transparent to, and compatable with the old STORE facility. Recognition by the new STORE facility of either the old or the new STORE data structure is facilitated by the additional ID string and checksum described in paragraph a) in section 7.3.3.1.

The STORE/RESTORE facility now provides a more convenient interface for manipulating multi-volume STORE volume sets.

STORE, when SHOW is specified, will addionally provide the list of files stored, by volume.

RESTORE STRATEGY

RESTORE accepts an arbitrary volume and a list of file names to be restored. By scanning the file names in the directory for preceding volumes, it determines the specific volumes to be mounted. The remaining files must, if valid reside on the current or following volumes. RESTORE scans through the current and sequential volumes until these files are obtained.

The operator is then asked to mount the predetermined preceding volumes to fulfill the user's demands. Files which span volumes may necessitate more volumes being mounted.

Obviously, the most efficient way to restore a few files from a large multi-volume set would be to mount the last volume of the set such that the precise volumes on which all the file reside could be determined. This results in a minimum number of mount operations.

6 INTERNAL INTERFACE CONSIDERATIONS

6.1 IMAGE

Since IMAGE directly utilizes the STORE/RESTORE facility, the above modifications admit to the requirement that minor changes be made to IMAGE in parallel with the effort on MPE.

C CONDENSE FACILITY

OVERVIEW

The Disc Condense Facility will allow users to reduce the amount of disc fragmentation which may exist on a System or Private Volume. The facility will be available as part of the VINIT subsystem.

The COND function of the VINIT subsystem is the mechanism whereby a user with sufficient capability can selectively, and on a volume-by-volume basis, remove the amount of space fragmentation which exists on a disc volume. The effect of a condense operation is to cause the free-space areas of a volume to be coalesced into larger blocks by the shifting of intervening in-use areas; the result of a condense operation is the creation of a pool of larger free-space areas which can satisfy a greater percentage of user disc-space requests. The user can determine the need for, and result of, a condense operation by use of the PFSPACE function of the VINIT subsystem.

While a condense operation is being performed all directory and disc-space operations not part of the condense itself will be suspended until the operation is complete. As a result, other users on the system are likely to be suspended as will further logon attempts until the condense operation has completed. In addition, the existance of any temporary disc areas associated with a job session may decrease the overall efficiency of a condense operation. For these reasons, condense operations should be performed at times when no other users can be expected to be on the system.

The condense operation will not have a fail-safe capability. That is, should the system experience a failure or the operation otherwise terminate abnormally the volume or directory may be left in a dirty state. This may cause the loss of the volume set or the reload of the system depending on the type of volume involved.

CAPABILITIES

A user of the VINIT subsystem must have operator capability; a user wishing to perform a condense operation must have in addition either system manager or system supervisor capability. Greater capability is required because of the potential effect on other users of the system, i.e. possible suspension of processing until the operation is complete.

OPERATION

The VINIT subsystem is invoked by the :VINIT command described in the Private volumes portion of this document. Once in the subsystem the user enters an appropriate COND command for each volume that is to be condensed. A condense operation can be performed on either a system volume or a Private volume; serial volumes cannot be condensed. Once a condense operation has begun, the VINIT subsystem performing the operation cannot be aborted until the operation has completed.

The COND function command has the following syntax:

```
          ;SIZE=n
COND ldn (        )
          ;ALL
```

The optional parameter pertains to the desired effectiveness of the condense operation. For reasons of user convenience, a condense will not normally attempt to coalesce all free areas because the more areas that are to be combined the longer the operation will take. Thus, unless the optional parameter is specified, only free areas of 50 sectors or greater in size will be considered by the condense algorithm. As this less time consuming condense will result in less than optimal tree-space reclamation, the user is given the power to override the default size limitation by specifying either that all free areas are to be combined if possible (ALL option) or that only areas of a specified size or greater are to be condensed (SIZE option). The SIZE option can also be used to further reduce the time of the operation by specifying a size factor greater than the default factor.

A condense operation can be performed on either System or Private volumes. In the case of a Private volume, the master volume of the set of which the volume to be condensed is a member must be mounted at the time of the condense operation.

DISCUSSION

Due to the nature of the applications being run on the HP3000 by many of the Users, HP is required to continually re-evaluate the requirements for more superior backup and recovery techniques to preserve the integrity of both System and User data environments on Disc on the off-chance that a System Error may occur. The new ennancements of MPE IIB provide much more capability in this direction. All new techniques have been discussed in detail earlier in this document, but it is worthwhile to re-iterate both the new and current funtions to demonstrate the impact on backup and recovery.

The release of User Logging is an important step in providing a method to assist both subsystems and applications to recover data in the event of a system error. This topic, presented earlier, is well defined and will not be discussed further in this section. The primary conentration will be on the remaining System fuctions which will be available to manipulate data on Disc, how they relate to one another, and the attributes of the data which they transfer.

.2 DISC STRUCTURE AND DEFINITIONS

With the advent of Private Volumes, disc devices on a system will be allocated to either of two groups:

- System Domain : all disc drives which are considered by system as permanently mounted and which cannot be used to dynamically mount and dismount private volumes. This includes the System Disc.

- User Domain : (Non-System Domain) consists of those disc drives suitable for use as private volumes (and/or serial discs) which may be mounted and dismounted as required.

Data residing on discs will be regarded as belonging to two groups:

- System Data : data, code, and files which constitute the operating system MPE IIB and reside in the System Domain.

- User Files : all other data/program files which belong to a user defined under the accounting structure and which may reside in either the System or User Domain.

Three types of "offline" storage media will be available:

- Magnetic Tape : as currently defined

- Serial Disc : defined to be a removeable disc which is accessed in a serial mode similar to Magnetic tape. It is not a private volume, but when accessed it must reside on a disc drive belonging to the User Domain.

- PV Disc : A removeable disc which is formatted and accessed by the Private Volumes facility. When accessed it must reside on a disc drive belonging to the User Domain.

.3 FUNCTIONS AVAILABLE

FUNCTION	SOURCE	DATA TYPE	DESTINATION	MODE
:STORE	System Domain User Domain	User Files	Magnetic Tape Serial Disc	On-line
:SYSDUMP	System Domain	System Data User Files	Magnetic Tape Serial Disc	On-line
:RESTORE	Magnetic Tape Serial Disc (from :STORE or :SYSDUMP)	User Files	System Domain User Domain	On-line
INITIAL	Magnetic Tape Serial Disc (from :SYSDUMP)	System Data User Files	System Domain	"On-line"
VINIT (>COPY)	PV Disc	"Volume copy"	PV Disc	On-line
SADUTIL (SAVE)	System Domain	User Files	Magnetic Tape	Off-line
RECOVER2	Magnetic Tape (from SADUTIL)	User Files	System Domain	On-line

.4 CONFIGURATION DEPENDENCIES

The funtional capabilities listed above are influenced by the disc configuration of any given System. The number of disc drives in the User (Non-System) Domain have a direct impact on the different ways by which Backup and Recovery may be facilitated. For example, the COPY funtion in the VINIT subsystem requires two removeable disc packs of the same type in order to do a volume copy. For ease of thinking, assume that all discs are of the same type in the following discussion concerning the exceptions which apply due to various configurations

User Domain : 2 or more discs

No problem. All functions available.

User Domain : 1 disc only

:STORE User Domain to Serial Disc NOT possible

:RESTORE Serial Disc to User Domain NOT possible

VIRIT COPY NOT possible.

User Domain : not configured

All operations to storage media Serial Disc or PV Disc are unavailable. In other words, the identical capability as is currently available under MPE II.